2007-03

Client location in 802.11 networks

Bangalan, Rolan T.

Monterey, California. Naval Postgraduate School

THESIS

CLIENT LOCATION IN 802.11 NETWORKS

by

Tuan Q. Dang
Rolan T. Bangalan

March 2007

Thesis Advisor:	Gurminder Singh
Co-Advisor:	Arijit Das

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2007	**3. REPORT TYPE AND DATES COVERED** Masters Thesis
4. TITLE AND SUBTITLE Client Location in 802.11 Networks		**5. FUNDING NUMBERS**
6. AUTHOR(S) Dang, Tuan Q. Bangalan, Rolan T.		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited		**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Location awareness is invaluable to the military commander. Any application that can accurately deliver this service is highly desirable. Being able to extract accurate distances is the first step towards developing a proposed 802.11 local area positioning system. This thesis explores a number of different methods of using 802.11 to capture physical distance separation between mobile stations. The first method of measuring distance, involves using 802.11 round trip signal times. Round trip signal times are determined from a transmitter to a receiver and back, and are used with the speed of light to measure distance between the nodes. Another method of using 802.11 to measure distance involves using signal strength measurements and a client-server arrangement. Distances can be extracted by extrapolating through a range of signal strength measurements. Because signal strength is a measurement of power, its behavior is governed by the inverse-square law. If environmental variables, such as humidity and RF interference do not significantly change, a line graph of signal strength measurements versus distance can be used to determine positions under these constant conditions.

14. SUBJECT TERMS IEEE 802.11, client location, signal strength, beacon round-trip time, packet round-trip time			**15. NUMBER OF PAGES** 91
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

CLIENT LOCATION IN 802.11 NETWORKS

Tuan Q. Dang
Lieutenant Commander, United States Navy
B.S., University of California, Berkeley, 1995

Rolan T. Bangalan
Lieutenant, United States Navy
B.S., San Diego State University, 1997
B.S., Polytechnic University of the Philippines, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2007

Author:	Tuan Q. Dang Rolan T. Bangalan
Approved by:	Dr. Gurminder Singh Thesis Advisor
	Arijit Das Co-Advisor
	Dr. Peter J. Denning Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Location awareness is invaluable to the military commander. Any application that can accurately deliver this service is highly desirable. Being able to extract accurate distances is the first step towards developing a proposed 802.11 local area positioning system. This thesis explores a number of different methods of using 802.11 to capture physical distance separation between mobile stations. The first method of measuring distance, involves using 802.11 round trip signal times. Round trip signal times are determined from a transmitter to a receiver and back, and are used with the speed of light to measure distance between the nodes. Another method of using 802.11 to measure distance involves using signal strength measurements and a client-server arrangement. Distances can be extracted by extrapolating through a range of signal strength measurements. Because signal strength is a measurement of power, its behavior is governed by the inverse-square law. If environmental variables, such as humidity and RF interference do not significantly change, a line graph of signal strength measurements versus distance can be used to determine positions under these constant conditions.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....................1
 A. BACKGROUND....................1
 B. PURPOSE....................2
 C. RESEARCH QUESTIONS....................4
 D. SCOPE AND METHODOLOGY....................4
 E. ORGANIZATION OF THE THESIS....................5

II. THE 802.11 PROTOCOL AND WI-FI SERVICES....................7
 A. THE 802.11 PROTOCOL....................7
 B. WI-FI CERTIFICATION....................9
 C. SERVICE SETS....................9
 D. WI-FI SERVICES....................10
 E. FRAME COLLISIONS....................11
 F. CSMA-CA....................12
 G. RTS-CTS CONTROL FRAMES....................13
 H. INTERFERENCE IN THE ISM RANGE....................14
 I. 802.11 CHANNELS....................14

III. DETERMINING DISTANCES USING PACKET ROUND TRIP TIMES.........17
 A. INTRODUCTION....................17
 B. NETWORKING SOCKETS....................18
 C. CLIENT-SERVER MODEL....................19
 D. SOCKET TYPES....................19
 E. PACKET PROPAGATION TIMES....................20
 F. DATA AND ANALYSIS....................23

IV. DETERMINING DISTANCES USING BEACON ROUND TRIP TIMES........29
 A. INTRODUCTION....................29
 B. PACKET ENCAPSULATION....................29
 C. THE BEACON FRAME....................30
 D. LINUX ACCESS POINT....................31
 E. NETGEAR NETWORK INTERFACE CARD & THE PRISM 54 PROJECT....................32
 F. LINUX ACCESS POINT SETUP....................33
 G. THE NETWORK DRIVER....................34
 H. NETWORK INTERFACE CARD CHIPSET....................34
 I. FIRMWARE....................35
 J. LINUX LOADABLE KERNEL DRIVERS AND COMPILING THE KERNEL....................37
 K. EXTERNAL DRIVER MODULES....................39
 L. LOADING THE MODULES....................39
 M. WIRELESS CONFIGURATIONS....................39
 N. NETWORK CONFIGURATIONS....................41
 O. PRISM 54 BEACON ROUTINE....................41

P. CALCLULATING DISTANCES USING LAYER TWO FRAMES........43

V. SIGNAL STRENGTH........45
A. SIGNAL STRENGTH UNITS OF MEASURE........45
B. “INVERSE-SQUARE” RELATION........46
C. METHODOLOGY........47
D. RESULTS........50

VI. CONCLUSIONS........55
A. ROUND TRIP TIMES........55
B. SIGNAL STRENGTHS........56

APPENDIX A. CLIENT SERVER SOURCE CODE........61
A. CLIENT ROUTINE........61
B. SERVER ROUTINE........64

APPENDIX B. RF POWER CONVERSION TABLE........67

LIST OF REFERENCES........73

INITIAL DISTRIBUTION LIST........77

LIST OF FIGURES

Figure 1. Trilateration Schematic. 3
Figure 2. Common 802.11 Protocols 8
Figure 3. Basic Service Set 10
Figure 4. Extended Service Set 10
Figure 5. Inaudible Transmitters 12
Figure 6. Neighboring Transmitters 12
Figure 7. 802.11b/g Eleven Channels with 5 MHz Separation 15
Figure 8. 802.11b/g Three Channels with 30 MHz Separation 15
Figure 9. Round Trip Time Schematic 17
Figure 10. OSI model 18
Figure 11. UDP Client/Server Socket Interaction. 20
Figure 12. Sample run number one at 158 meters 24
Figure 13. Sample run number two at 158 meters 24
Figure 14. Sample run number one at 158 meters with filtering 25
Figure 15. Sample run number two at 158 meters with filtering 26
Figure 16. Sample run number one at 83 meters with filtering 27
Figure 17. Sample run number two at 83 meters with filtering 27
Figure 18. Data Encapsulation 29
Figure 19. Beacon frame embedded in MAC layer 31
Figure 20. Netgear WG511 32
Figure 21. Intersil Prism GT Duette ISL3890 Chipset Firmware Add-On 37
Figure 22. make xconfig Configuration Utility 38
Figure 23. ifconfig Script File 40
Figure 24. iwconfig Utility 40
Figure 25. Signal Strength Units of Measure 46
Figure 26. Signal Strength (dBm) vs Distance (ft) 47
Figure 27. Signal Strength Measured at 25 ft Intervals from Access Point 48
Figure 28. Linux’s iwconfig 49
Figure 29. Signal strength measurements 51
Figure 30. Signal strength versus distance, 25 to 275 ft 51
Figure 31. RSSI at distances less than 10 meters from access point 52
Figure 32. Signal strength versus distance, 1 to 10 meters 53
Figure 33. Directional Antenna. 58
Figure 34. Omni-directional antenna 58
Figure 35. Decreasing antenna beam width. 59

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I (Tuan Dang) would like to thank Rolan, who I consider one of the hardest working individuals I know. Enjoy San Diego, you've earned it! Also, to my lovely wife, who I consider one of the most patient and understanding. No more late night homework assignments!

I (Rolan Bangalan) want to express my deepest gratitude to my lovely wife, Ning, for being a constant source of inspiration throughout my military career, especially while stationed here at NPS. I would also like to thank my children for their understanding, their willingness to sacrifice play time and vacations, and their constantly putting up with an exceptionally grouchy father. In addition, I am grateful to Tuan for his tremendous efforts in getting this thesis completed.

Lastly, both authors would like to express their appreciation for the valuable assistance given by Professor Gurminder Singh and Arijit Das. If not for their guidance, this project would not be possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Positioning systems in general are often categorized into three groups: global systems, wide-area systems that rely on cellular technologies and local area 802.11 positioning system. The Global Positioning System (GPS) for example, is a global system. It is a satellite based navigation system that provides accurate and reliable positioning information with world wide coverage. GPS however is unreliable in many urban areas, where line-of-sight access to satellites is often obstructed by satellite signal impenetrable structures such as large buildings. For indoor positioning, GPS serves little value. Additionally, separate hardware is required to enable GPS services. In areas where GPS is ineffective, wireless positional systems that use 802.11 protocols may be used to deliver positioning services. Positioning is common for a number of wide area systems, such as systems using cellular technologies; however the high cost of setting up a cellular infrastructure and the inaccessibility to proprietary cellular system codes makes is impractical for many wireless positioning applications. This thesis will focus on local area 802.11 positional systems.

Location awareness is invaluable to the military commander and any application that can accurately deliver this service is highly desirable. Positioning and tracking systems provide military commanders an important tool to plan, coordinate, and assess tactical operations. There are a number of different military applications of 802.11 based positional systems. A wireless enabled device could be connected to a Voice over IP (VoIP) phone to provide emergency distress location services. 802.11 could also be used to map out an area of operations to provide location services and directions as tactical waypoints. With radio frequency identification (RFID) technology, 802.11 could be used to track people and equipment for mission planning. The positioning system could be embedded within a wireless mesh network to extend range through backhaul connections. For example, a couple of root nodes are connected to the primary network or directly to the Internet. The mesh clients are connected to rest of the network, thus extending positioning information to servers that can process collected positional data.

Currently, there are a number of different wireless positional systems that are available, each of which use different 802.11 methods to provide positioning services. For example, Skyhook Wireless is a company that develops software which determines positioning information using a nationwide network of known Wi-Fi access points. Skyhook enabled devices collect the MAC address identification for every access point in its range, and compares the identification to a database that contains known locations for these access points. Using these known locations, Skyhook determines location based on its relative position to the known access points. These developers claim positional accuracy to within 20 to 40 meters, particularly in metropolitan areas where there is a large network of deployed wireless access points which are stationed close together.

B. PURPOSE

Being able to extract accurate distances is the first step towards developing a proposed 802.11 local area positioning system. This thesis explores a number of different methods of using 802.11 to capture physical distance separation between mobile stations. The first method of measuring distance, involves using 802.11 round trip signal times (RTT). RTT is determined from a transmitter to a receiver and back, and is used with the speed of light to measure distance between the nodes. The distance can then be imported into a trilateration algorithm to calculate positional information. For example, from Figure 1 below, relative location of a mobile client (B) can be accurately determined using a two dimensional grid. Three stationary wireless access points (P1, P2, and P3) determine respective distances (r1, r2, and r3) to the mobile client. Knowing r1 limits the possible locations around the circumference of the P1 circle. Knowing r1 and r2, narrows the possible locations down to either point A or point B. Knowing r1, r2 and r3, it is possible then to trilaterate the position of the mobile client to point B.

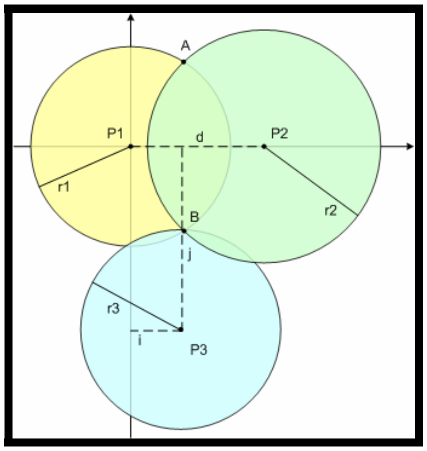


Figure 1. Trilateration Schematic.

Placing this grid over an x-y Cartesian coordinate system, relative positions can be further defined using P1 as an anchor, located for example at position (0,0) on the x, y coordinate plane. Once trilateration is accomplished, the distance from P1 to B is equal to r1, which then could be separated into x and y components.

An 802.11 access point can be used as a source for the RTT signal. For wireless local area networks (LANs), the beacon frame provides management functionality used to establish and maintain communications between the access point and a wireless station. Generally, the beacon intervals are set at about 100 ms, but the intervals can be controlled. By inserting a timer routine just prior to the transmission of the frame and immediately after the return of a response to the beacon frame, RTTs can be calculated. Distance between the nodes can then be found using pre-determined system processing times and theoretical propagation times for the signal.

Another method of using 802.11 to measure distance involves using signal strength measurements and a client-server arrangement. Distances can be extracted by extrapolating through a range of signal strength measurements. Because signal strength is a measurement of power, its behavior is governed by the inverse-square law. If environmental variables, such as humidity and RF interference do not significantly change, a line graph of signal strength measurements versus distance can be used to determine positions under these constant conditions.

C. RESEARCH QUESTIONS

1. Is it possible to take advantage of 802.11 capabilities in developing a system to capture physical distance separations between wireless nodes?
2. Can distances obtained through 802.11 methods be used with trilateration to accurately and precisely determine positional information?
3. Can determining the round trip time a wireless signal sent from wireless stations effectively be used to determine distance?
4. What is the relative impact of system processing time on round trip time calculations? Are the differences significantly large enough that it negatively impacts 802.11 distance measurements?
5. What are the statistical spread of round trip time propagation delay between the transmitter and receiver? Are the spread large enough that they prevent accurate determination of distances?
6. Are the processing times for signals that originate higher in the protocol stack significantly large enough that it prevents accurate distance measurements using round trip times?
7. How does the time it takes for a network interface card to modulate a transmission and reception signal contribute to system processing times?
8. How does 802.11 round trip time compare with signal strengths in determining distances?
9. What are the spatial limitations of 802.11 positioning systems?
10. What are the environmental variables that have the greatest impact on 802.11 signals?

D. SCOPE AND METHODOLOGY

The first step towards developing a wireless positioning system involves using 802.11 to capture distance values between wireless stations. One method that is investigated uses RTTs for 802.11 packets. Network packet signaling involves interprocess communications through sockets. The client, process one, establishes a connection to the server, process two, through the socket. A timer mechanism, such as Java's nanotimer API, is used to measure how long it takes for a packet to complete a round trip from the client to server and back. Using the general laws of RF propagation over the air, it is expected that distances can be determined.

Using network packets however adds an enormous overhead cost in terms of how long it takes to process the signal as it moves through the protocol stack. If the processing times are large enough, accurate distance measurements may be overwhelmed by the large processing time. An alternative method of using 802.11 to measure distance is also investigated. In this method, layer two beacon frames are used. One practical way of automating this setup involves using an 802.11 access point and establishing an infrastructure network. In creating an infrastructure network, the periodic beacon transmissions from the access point may be used as a source for the round trip signals. For this thesis a mobile laptop running Linux Fedora Core 4 was configured to serve as an access point.

Using signal strengths may also be used to measure distance. The laws of RF propagation, such as the inverse-square law for power can be used to extrapolate a finite set of signal strength measurements. These extrapolated values can then be used to predict distances.

E. ORGANIZATION OF THE THESIS

The chapters in this thesis are arranged according to the following topics. Chapter II is an overview of basic wireless concepts that are important to the work involved with this thesis. The chapter is not meant to be comprehensive, but rather it touches upon key points that are relevant to this paper. Chapter III explores the use of packet round trip times, using Java sockets, as a method of determining position. The results of this chapter establishes the foundation for Chapter IV which discusses the possibility of using automated 802.11 layer two beacon transmissions from a wireless access point as the signaling mechanism for round-trip time calculations. This chapter will also detail the setup and configuration of a Linux based wireless access point. Chapter V discusses a method of positioning using wireless signal strengths. Chapter VI completes this thesis with overarching conclusions and recommendations for future work.

II. THE 802.11 PROTOCOL AND WI-FI SERVICES

A. THE 802.11 PROTOCOL

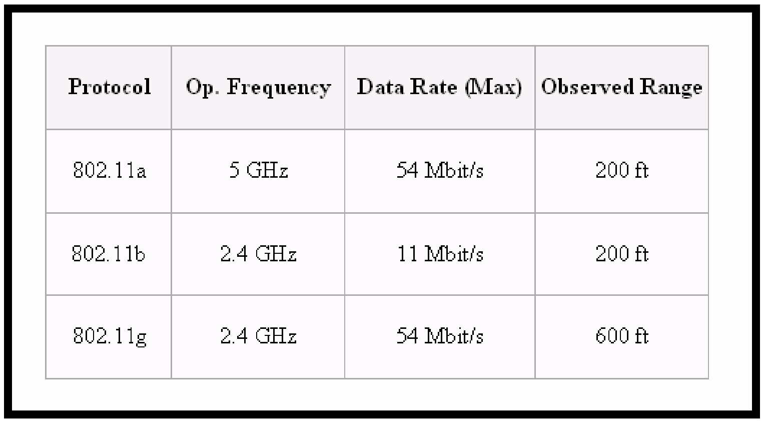
A discussion on certain details of the 802.11 protocol and Wi-Fi services is necessary to understand the methodologies used throughout this thesis. This chapter provides a glance at certain aspects of the 802.11 protocol and Wi-Fi services that are important to this thesis and is not meant to be a complete report on the operation of the 802.11 protocols and Wi-Fi services. For further details on both standards, the sources located in the appendix may be referred to for more specific information.

Wireless LAN technology is dominated by 802.11 protocols, which refer to a group of specifications that was approved by the IEEE in 1997. These specifications are important because they establish the radio standards necessary for over the air communications between wireless nodes and access point base stations, and communications between different wireless nodes.

802.11 is a link layer protocol, which is responsible for transferring data between adjacent nodes. The link layer establishes the functionality of sending data frames as well as the procedural arrangements to establish communications between the nodes. The link layer also provides error detection and correction mechanisms to maintain the integrity of the data frames.

Because link layers are only responsible for getting a data frame from a particular source to an adjacent destination, link layer addressing is extremely simple. Complicated routing tables are not necessary to move data frames because the link layer is not responsible for sending information to more than one place. It just has to establish communications with a single destination node. Communication with this single destination node however can occur over multiple channels because 802.11 also defines multiple-access through the Media Access Control (MAC) sub layer. The MAC sub layer allows communication to take place within a multipoint topology by implementing addressing and channel access controls. This allows nodes that are physically connected to same access point to share resources without the need for complicated routing mechanisms. 802.11 also serves as a physical layer standard, because it implements the mechanisms for transfer of data bits through radio transceivers that connect network nodes.

Since its inception in 1997, the 802.11 standard has added a number of major improvements. The most significant modifications occurred in the middle of 2003 with the development of the 802.11g standard. Changes included a higher data rate, medium access processes, and modulation techniques. 802.11g extends the capabilities of the 802.11b protocol and their technologies are compatible because they both operate in the 2.4 GHz range. This means that 802.11b and 802.11g devices can operate with one another however the performance limitations are restricted by 802.11b data rates. 802.11g is able to attain higher rates because it uses Orthogonal Frequency-Division Multiplexing (OFDM) modulation, the same method found in the 802.11a protocol. In January 2004 a committee was formed to address required standards for 802.11n amendments, which included proposals for maximum data rates approaching 540Mbs, a rate ten times faster than the 802.11g standard. It is expected that the 802.11n standard will be fully ratified by the end of 2007. Table 1 below lists typical performance characteristics for three commonly found 802.11 protocols.



Protocol	Op. Frequency	Data Rate (Max)	Observed Range
802.11a	5 GHz	54 Mbit/s	200 ft
802.11b	2.4 GHz	11 Mbit/s	200 ft
802.11g	2.4 GHz	54 Mbit/s	600 ft

Figure 2. Common 802.11 Protocols

The 802.11 specifications operate over the unlicensed band portion of the radio spectrum, which is reserved internationally for industrial, scientific and medical (ISM) non-commercial uses. Recently this portion of the electromagnetic field RF field has been widely used in a number of different license free applications such as Wi-Fi, Bluetooth and wireless telephone sets over the 2.4 GHz and 5 GHz bands. However different countries may have slight modifications to these operational communications band. For example, in the United States the FCC is the regulatory body responsible for establishing the rules for use of wireless communications over these RF bands.

B. WI-FI CERTIFICATION

802.11 standards do not address Wi-Fi certifications. Wi-Fi certification is granted to those wireless equipment that meet interoperability testing requirements as defined by the Wi-Fi Alliance (http://www.wi-fi.org/index.php). Certification ensures that equipment that are manufactured by different vendors are able to work together. Therefore, wireless equipment that adheres to 802.11 standards may or may not be Wi-Fi certified. For the purpose of this thesis, references to 802.11 technologies infer Wi-Fi compliance.

C. SERVICE SETS

The typical Wi-Fi arrangement includes at least two nodes. The nodes may be all wireless clients talking to each other over ad hoc connections or it may include access points. The nodes that establish communications directly with each other forms a Basic Service Set (BSS), Figure 3, and is recognized by an identification code called the BSSID. The BSSID is derived from the node’s MAC address. The BSS can be either an infrastructure based or ad-hoc based BSS. The infrastructure BSS (InfBSS) relies on access points as communication switches. Ad-hoc based BSS, or Independent BSS (IBSS) are comprised solely of wireless nodes that communicate peer to peer.

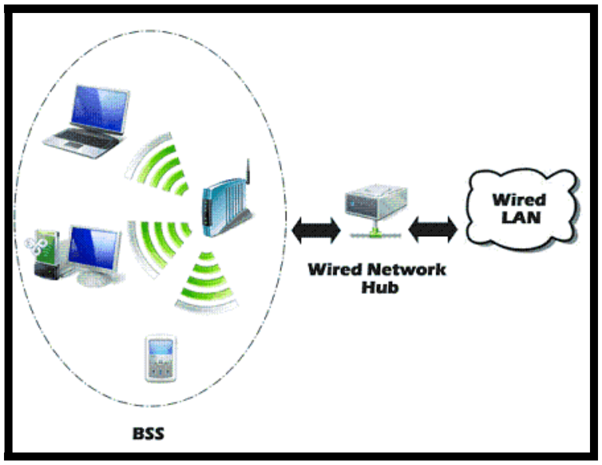


Figure 3. Basic Service Set

Extended Service Sets (ESS) are used to establish communications between nodes from one BSS to another BSS, Figure 4. For the scope of this thesis, communications occur over the peer to peer BSS domain.

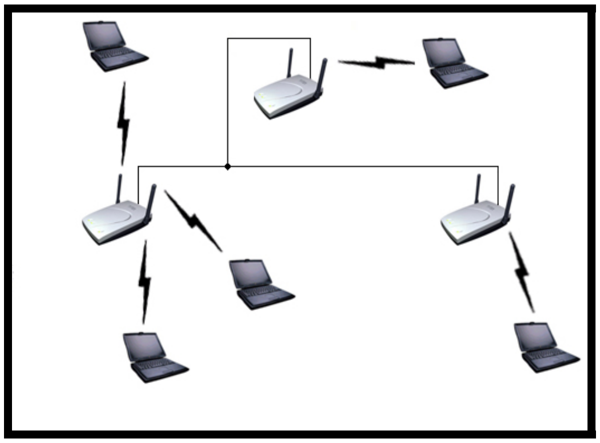


Figure 4. Extended Service Set

D. WI-FI SERVICES

Wi-Fi standards establish nine basic management services: Association, Disassociation, and Re-association, Authentication, De-authentication and Privacy, Packet Delivery and Distribution, and Integration. The management services are responsible for the initial establishment, maintenance and security of communications. This thesis however primarily focuses on the association and packet delivery services.

The MAC layer is responsible for the rules that govern association services. The rules establish the procedures necessary for discovery and connection of a node to an access point or another peer node. Before association takes place a wireless node is constantly scanning for wireless networks to connect. The node is aware of available networks through beacon advertisements. Beacons are broadcast frames used to advertise the location of nodes. The frames contain the Service Set Identifier (SSID) which is a 32 bit text string used as a unique identifier for both the BSSs and ESSs. The beacons are sent periodically according to intervals established by the user; however 100 ms is usually the default setting. The periodic transmissions ensure that nodes are able to synchronize with one another for functions such as association and power management operations. Association to a BSS or IBSS occurs through a three phase process as outlined by the IEEE. Phase 1 is the un-associated state, phase 2 is authenticated but un-associated state and phase 3 is the associated phase. In order to authenticate the node first sends an authentication request. If security is enabled, the access point or peer node responds with a shared key challenge. The node enters phase 2 after it successfully satisfies the challenge request. Once authenticated the node sends an association request, which is a frame that includes information such as the SSID of the requested network and management information such as its supported data rate. The node is associated with the network once it receives an association acceptance frame and is then free to transmit data. Since this thesis involves open authentication (no authentication), phase 2 is not applicable.

E. FRAME COLLISIONS

Frames transmitted over the air are susceptible to collisions, similar to frames transmitted over the wire. Because it is impractical to implement a collision detection protocol, such as that used in Ethernet, Wi-Fi standards establish an avoidance protocol to minimize loss of frames due to collisions. Collision detection is not used for over the air transmissions because of two particular situations.

The first situation, Figure 5, involves a transmitter that is incapable of hearing a second transmitter, because of extended range. Both transmitters however are in range of the receiver, but collision detection is not possible because the first transmitter does not know when the other transmitter is sending frames.

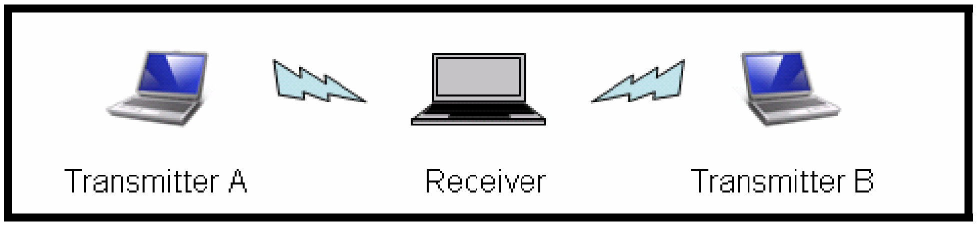


Figure 5. Inaudible Transmitters

A second situation involves a topology similar to that in Figure 6 below. The Transmitter-Receiver pairs A and B are in range of one another, the Transmitters A and B are in range of one another but the Transmitter A and Receiver B (and Transmitter B and Receiver A) are not in range of each other. When Transmitter A or B attempts to send frames when the other Transmitter is sending frames, carrier sense prevents simultaneous transmission, even though the intent is to send frames to different destinations.

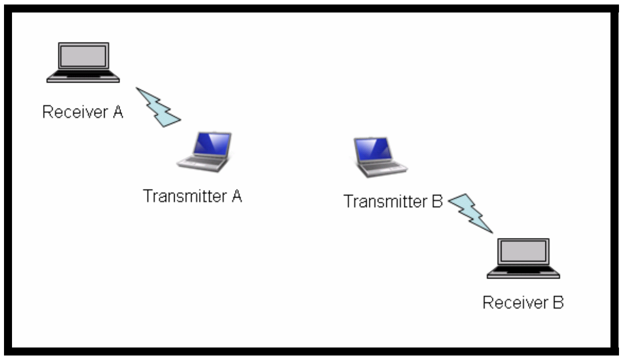


Figure 6. Neighboring Transmitters

F. CSMA-CA

Wi-Fi uses a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol because collision detection is not practical for most Wi-Fi hardware. This is because most Wi-Fi network cards have only a single antennae and full duplex operation is not possible. The antennae can either send or receive at a given moment in time, but it cannot do both simultaneously. Full-duplex cards are available but extremely expensive to manufacture and therefore collision avoidance protocols are more commonly found in Wi-Fi gear.

G. RTS-CTS CONTROL FRAMES

CSMA/CA is an 802.11 control service that is responsible for the delivery of frames between nodes. Before a node begins transmitting a frame, it listens for activity (carrier sense) to determine if a channel is available for sending data. An available channel indicates that the channel is clear of transmissions and that it has been reserved for transmission. This is achieved using Request To Send (RTS) and Clear To Send (CTS) frames that are sent between the source node, the destination node, and all audible nodes in range. Wi-Fi CSMA/CA protocols use RTS and CTS frames for collision management. RTS and CTS frames are 802.11 standards designed to address issues related to performance of wireless systems, such as the problem associated with neighboring transmitters. An RTS frame is sent from the source node to the destination node. The destination replies by broadcasting CTS frames, which are used to inform the source node that it is ready to receive data. Additionally the CTS frames are designed to instruct other audible nodes to wait after an established delay period before attempting to send frames. Transmission of data frames begins after the source node receives the CTS response. The receiver sends an acknowledgment to the source, indicating completion if error checking is successful. If the node senses that the medium is being occupied, it starts a timer that randomly establishes a delay before it again attempts to sense the medium for availability. During the RTS-CTS exchange, if frames are dropped or lost, the sequence begins again from the start after the expiration of a random delay time.

H. INTERFERENCE IN THE ISM RANGE

Operation of Wi-Fi in the ISM range introduces a number of problems related to RF interference since the ISM range includes a large number of different commercially available devices that ignore 802.11 protocols. In the 2.4 and 5 GHz range it is common for devices such as wireless telephones, microwave ovens, and other 802.11 devices operating on the same channel to interfere with the primary signal. Because Wi-Fi data packets are sent in accordance with collision avoidance protocols, delivery of data can only occur when the transmission medium is available. This means when one node is transmitting, a second node must wait until the channel is idle before it can begin transmission. If a noncompliant device is transmitting over a frequency being used by an 802.11 device, the 802.11 device may have to wait unnecessarily until the medium becomes available.

Another situation occurs when an 802.11 device is in the middle of transmitting data. Because a noncompliant wireless device does not have to operate in accordance with 802.11 rules, it does not adhere to collision avoidance protocols. If the noncompliant device begins transmission while an 802.11 device is transmitting, interference signals are introduced which can lead to dropped packets. If the packets arrive at the destination node, the data may be corrupted. If the source does not receive a successfully acknowledgement frame it continues to send packets which adds additional load to the network medium.

I. 802.11 CHANNELS

Splitting the network frequencies into channels alleviates the problem of interference between nearby 802.11 devices. For example, three channels, 1, 6, and 11 are used for the 802.11b/g standard in the United States. A channel is configured for the access point or an ad-hoc node. A node that wishes to establish communications automatically tunes its radio to the chosen channel to establish association. Tuning however depends of the strength of a signal. Given two access points that are in the vicinity of one another, and both are assigned the same channel, tuning will bind to the access point that has the higher signal strength.

Eleven channels, separated by 5 MHz, are actually assigned to the 802.11b/g standard. However because a typical 802.11b/g transmission uses 30 MHz of bandwidth, the number of available channels is effectively reduced to three channels in order to eliminate signal overlap, Figures 7 and 8. Fourteen channels are assigned to the 802.11a standard.

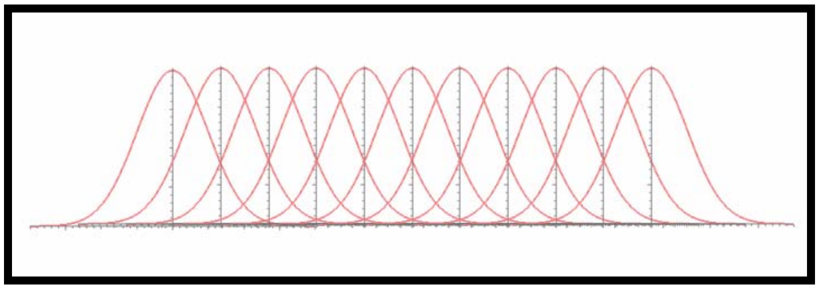


Figure 7. 802.11b/g Eleven Channels with 5 MHz Separation

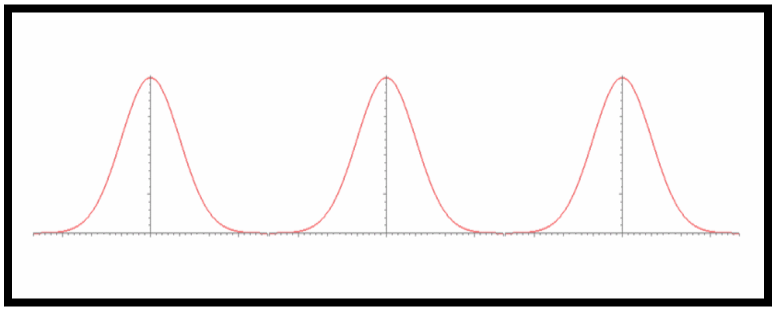


Figure 8. 802.11b/g Three Channels with 30 MHz Separation

THIS PAGE INTENTIONALLY LEFT BLANK

III. DETERMINING DISTANCES USING PACKET ROUND TRIP TIMES

A. INTRODUCTION

Experiments were conducted to determine the possibility of using round trip packet propagation times between two wireless nodes to compute distance. The setup involves a client that initiates a packet transmission and a server which captures the packet and re-routes the original packet back to the client, Figure 9. At the client, a timer API is used to measure total round trip time, which is then used along with the physical propagation property of RF transmissions over the air to extract processing time intrinsic to the given system configuration. Once an estimate of total processing time is obtained, it can then be used along with measured round trip times to compute distances as the nodes are moved at varying distances from one another.

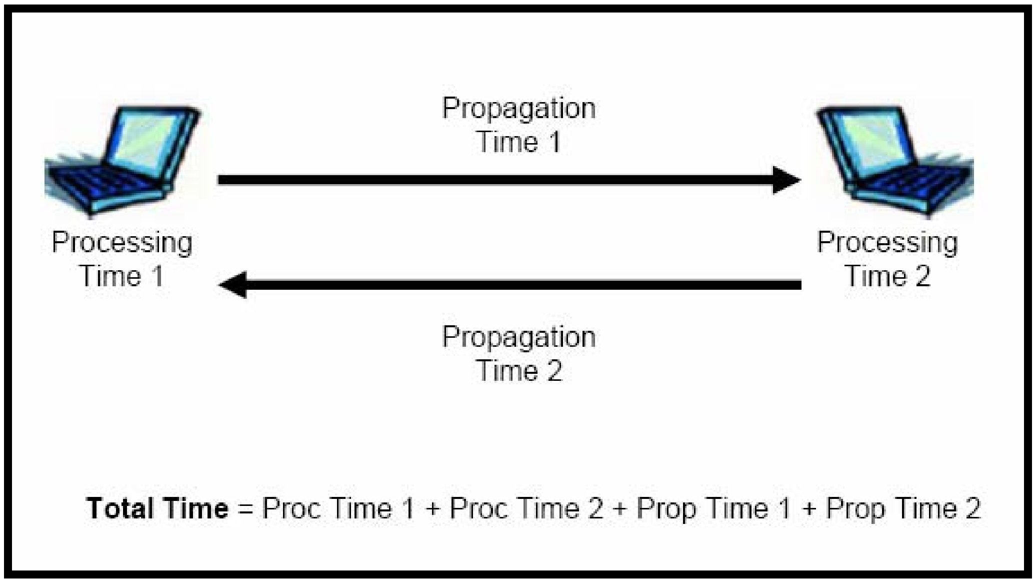


Figure 9. Round Trip Time Schematic

B. NETWORKING SOCKETS

A networking socket is a software endpoint that establishes bidirectional communication between a server program and one or more client programs. The socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program.

Networking sockets reside at the Session Layer of the OSI model, Figure 10.

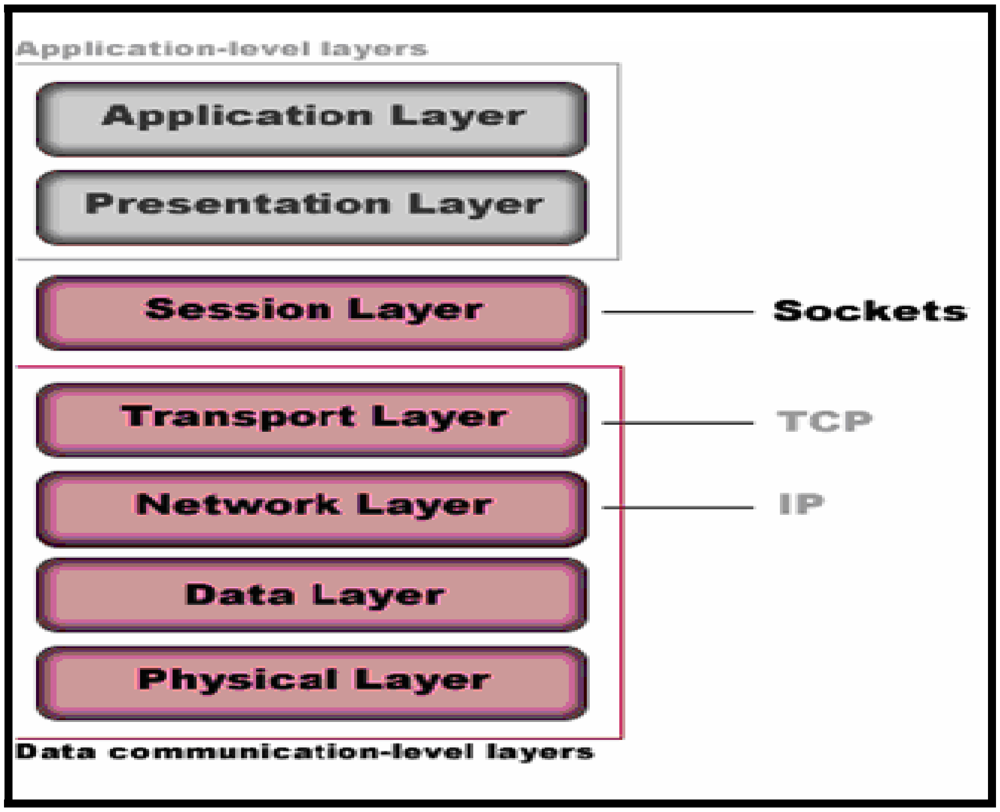


Figure 10. OSI model

The Session Layer is positioned between the application-oriented, upper layers and the real-time data communication, lower layers. The Session Layer provides services for managing and controlling data flow between two computers. As part of this layer, networking sockets provide an abstraction that hides the complexities of transmitting the bits and bytes of information.

C. CLIENT-SERVER MODEL

The client-server model provides a convenient way to connect processes that are physically separated. A server program typically provides resources to a network of client programs. Client programs send requests to the server program and the server program responds to each request. Most network communication uses the client-server model. The client and the server are the two processes which will be communicating with each other. The client requests for service provided by the server. Bear in mind that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

The procedures for establishing a connection are different for the client and the server, but involve the creation of a networking socket. Both the client and server create their own respective sockets.

On the client side, the following steps are taken; create a socket with the socket() system call, connect the socket to the address of the server using the connect() system call, and finally, send and receive data. On the server side, the following steps are taken; create a socket with the socket() system call, bind the socket to an address using the bind() system call, listen for connections with the listen() system call, accept a connection with the accept() system call, and finally, send and receive data .

D. SOCKET TYPES

The two widely used socket types are stream sockets and datagram sockets. Each uses its own communications protocol. Stream sockets use Transmission Control Protocol (TCP) and datagram sockets use User Datagram Protocol (UDP). Stream sockets use a reliable, full duplex, connection-oriented means of communications, while datagram sockets use unreliable, bidirectional data flow. Datagram sockets, because of its reliance on UDP, are referred to as connectionless sockets. When a networking socket is created, the program has to provide the service name, the protocol, and port number. Figure 11 shows how a client's network socket interacts with the server's network socket.

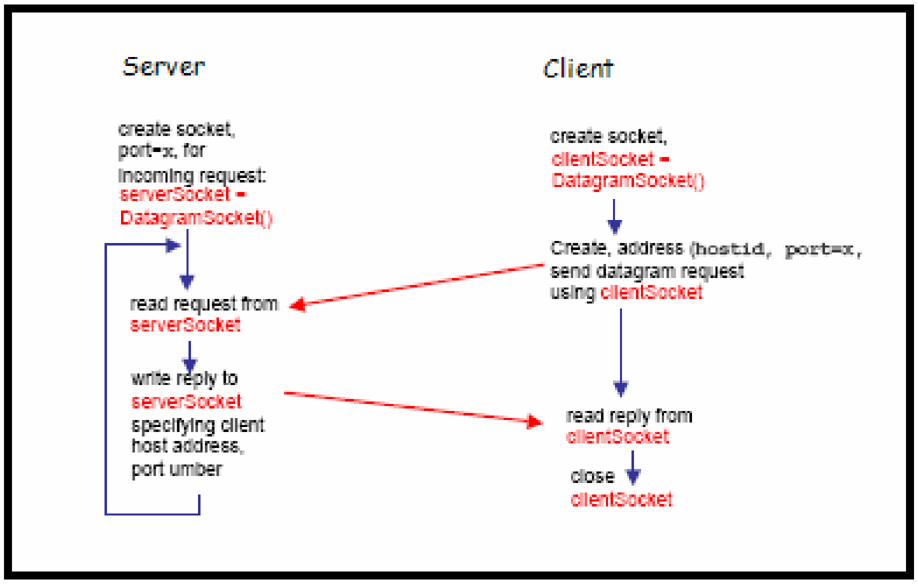

Server
create socket, port=x, for incoming request: serverSocket = DatagramSocket()
read request from serverSocket
write reply to serverSocket specifying client host address, port umber
Client
create socket, clientSocket = DatagramSocket()
Create, address (hostid, port=x, send datagram request using clientSocket
read reply from clientSocket
close clientSocket

Figure 11. UDP Client/Server Socket Interaction.

E. PACKET PROPAGATION TIMES

For this experiment, programming was performed in Java. Two Java classes were created, NanoServer and NanoClient. The NanoServer class designated one of the wireless nodes as a server and used port 9876 to wait for UDP packets from the client. As soon as the server received a packet from the client, it checked the payload to ensure that the UDP packet came from the client running the NanoClient class. The following is a fragment of the NanoServer class code, Appendix A.

```
class NanoServer {
    public static void main(String [] args) throws Exception {
        // Port Number
        DatagramSocket serverSocket = new DatagramSocket(9876);
        while (true) {
            …
        //Receive Packet
        DatagramPacket receivePacket = new
          DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
        ...
        sendData = entry.getBytes();
        // set to IP Address
        DatagramPacket sendPacket = new
          DatagramPacket (sendData, sendData.length,  IPAddress, port);
        serverSocket.send(sendPacket);   } } }
```

The following is a fragment of the NanoClient class code, Appendix A. The NanoClient class establishes a Datagram socket for the UDP Client-Server. It ensures minimal overhead during processing packets between the two nodes. Since Java provides a nanosecond counter, this feature was used in the algorithm. The method nanoTime( ) was started as soon as the UDP packet left the client. As soon as the UDP packet was received from the server, the nanoTime( ) method is invoked again. The difference was recorded in a data structure called Statistics object. The Statistics object provided histograms and calculation of the mean.

```
public class NanoClient{
    public static void main( String args[] ){
        SimpleStats ss = new SimpleStats( );
        Histogram hist = new Histogram ("Output", 0, 2600000,  100);
        for (int i=0; i <= 1000; i++){
        try {
                DatagramSocket clientSocket = new DatagramSocket();
                  ...
                // UDP IP address and Port Number
                DatagramPacket sendPacket =new
                  DatagramPacket(sendData, sendData.length,
                  IPAddress, 9876);
        // Send Packet
        clientSocket.send(sendPacket);
        long start = System.nanoTime();
        // Start Timer
        System.out.println (start);
        DatagramPacket receivePacket = new
          // UDP RECEIVE
          DatagramPacket(receiveData,
          receiveData.length);
          ...
        // Start Timer
        long stop = System.nanoTime();
        long difference = stop-start;
        System.out.println (difference); // Difference
```

Measurements were conducted in accordance with the following procedures:

1. Estimate Theoretical Propagation Time (T_{th_prop}) given a known distance between the client and server. This thesis uses ideal RF propagation of packets over the air to compute theoretical propagation times.

$$T_{th\_prop} = Prop\ 1 + Prop\ 2 = 2 * (3 * 10^8\ m/s) * distance$$

2. Measure Round Trip Time (RTT) using timer API routine. The measured RTT is equal to the combined total of processing and propagation times for the client and server.

$$RTT = Total\ Time = Proc\ 1 + Proc\ 2 + Prop\ 1 + Prop\ 2$$
$$RTT = Proc\ 1 + Proc\ 2 + T_{th\_prop}$$

3. Compute Total Processing Time (T_{proc}). T_{proc} does not change if the system configuration does not change. For example, T_{proc} will increase or decrease depending on whether a new process is started or a running process is stopped. T_{proc} is then used in future measurements, along with measured RTT at varying client server positions to compute distance.

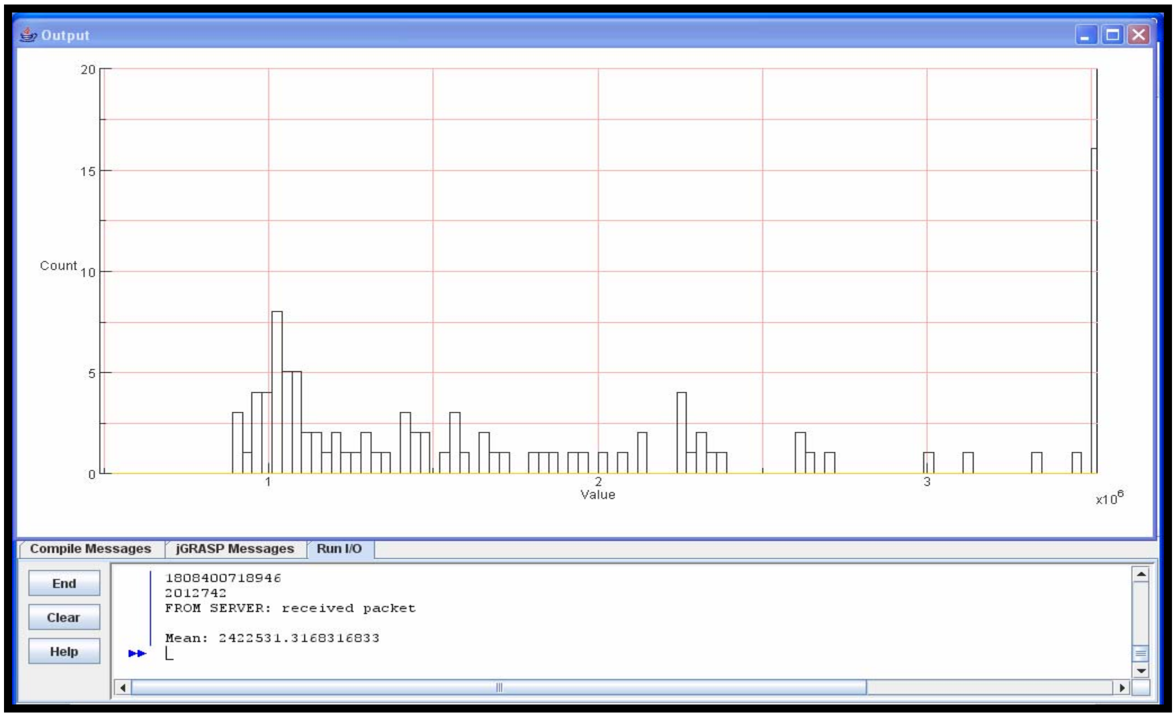
$$T_{proc} = Proc\ 1 + Proc\ 2$$
$$RTT = T_{proc} + T_{th\_prop}$$
$$T_{proc} = RTT - T_{th\_prop}$$

4. Assuming T_{proc} remains constant, determine the distance between wireless nodes, using measured RTTs.

$$T_{th\_prop} = RTT - T_{proc}$$
$$2 * (3 * 10^8\ m/s) * distance = RTT - T_{proc}$$
$$distance = (RTT - T_{proc}) / 2 * (3 * 10^8\ m/s)$$

F. DATA AND ANALYSIS

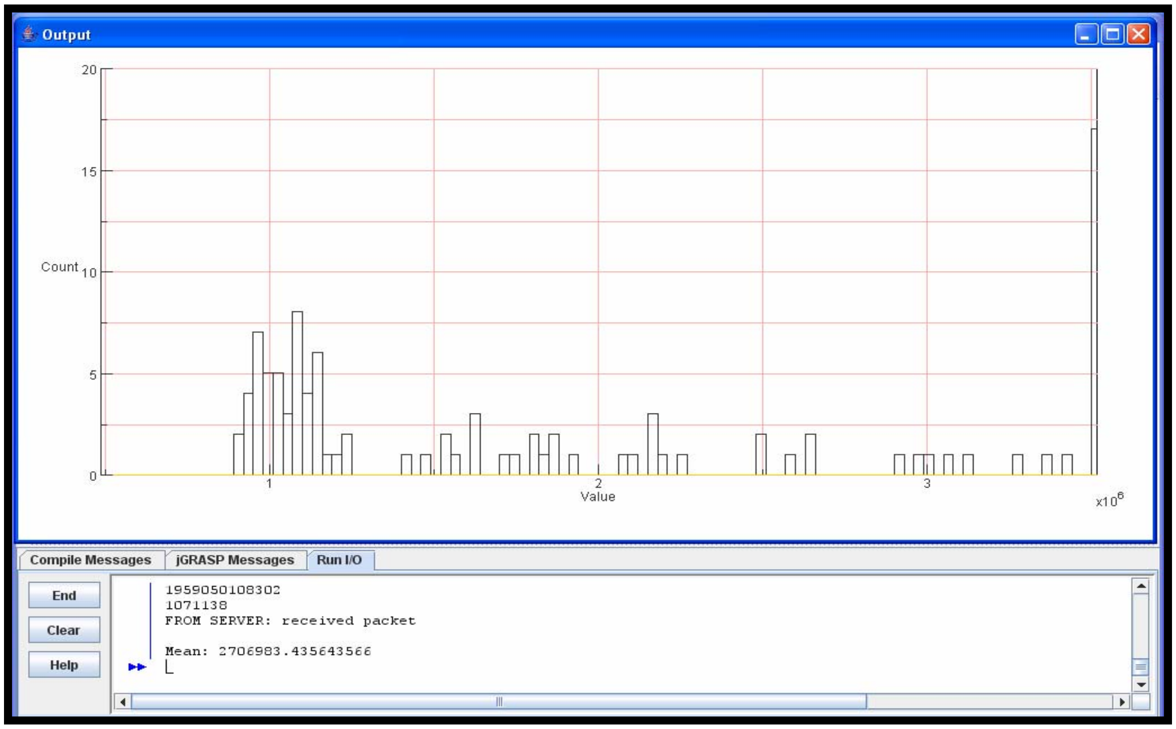
Figures 12 and 13 show the lowest and highest readings generated. For 30 executions of the Java application, the outcome was a mean of 2.6 ms.



```
1808400718946
2012742
FROM SERVER: received packet

Mean: 2422531.3168316833
```

Figure 12. Sample run number one at 158 meters

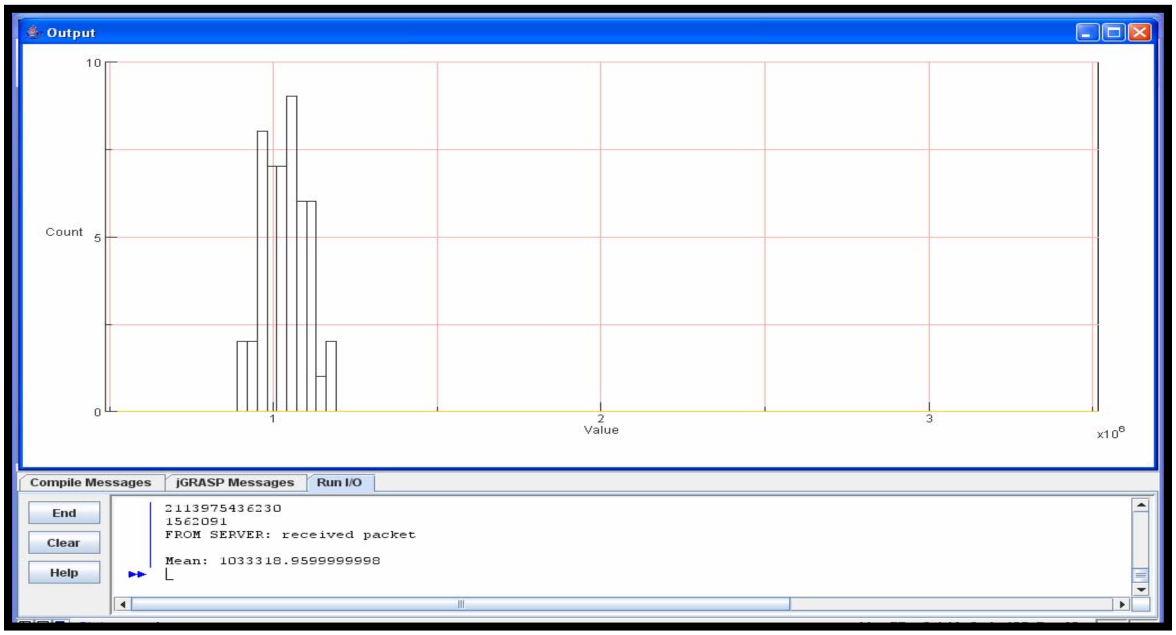


```
1959050108302
1071138
FROM SERVER: received packet

Mean: 2706983.435643566
```

Figure 13. Sample run number two at 158 meters

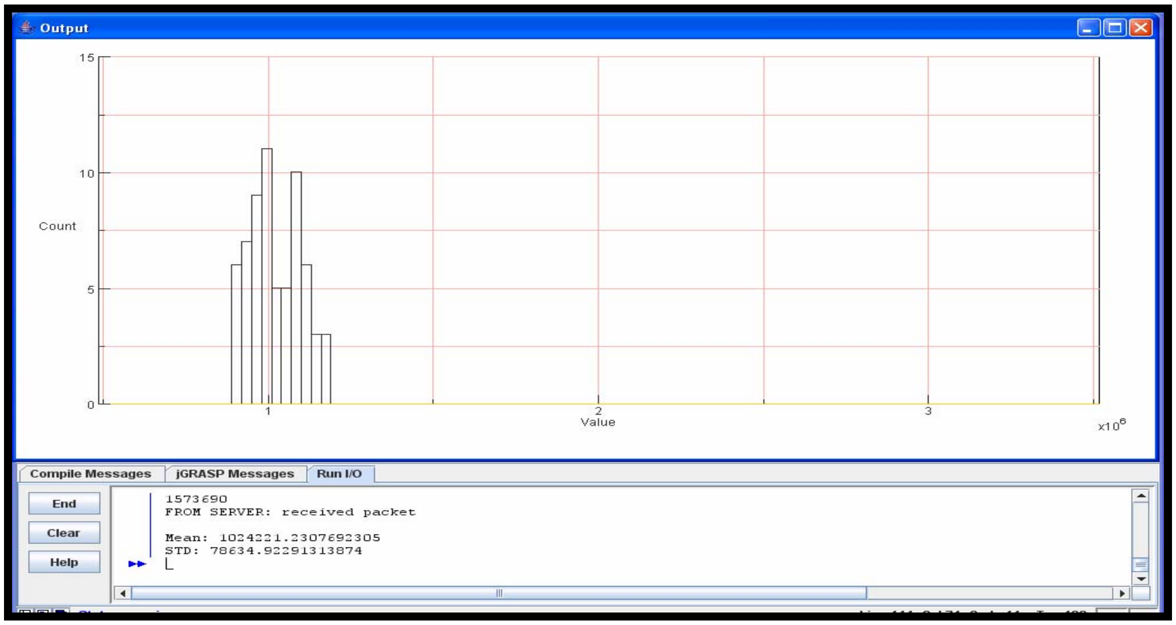
Since most of the data values clustered around 1.0 ms, a filter was applied to purge the outliers. Outliers are due to radio signals not taking the shortest path between the client and the server. For example, since the wireless card is omni-directional a signal may be initially sent away from the receiver, bounce off a reflector a number of times before reaching the intended target. Because of this effect, a signals propagation time is higher than an ideal situation where a signal takes a straight path from the transmitter to the receiver. With the filter incorporated in the Java code, readings were again taken, as seen in Figures 14 and 15.



```
2113975436230
1562091
FROM SERVER: received packet

Mean: 1033318.9599999998
```

Figure 14. Sample run number one at 158 meters with filtering



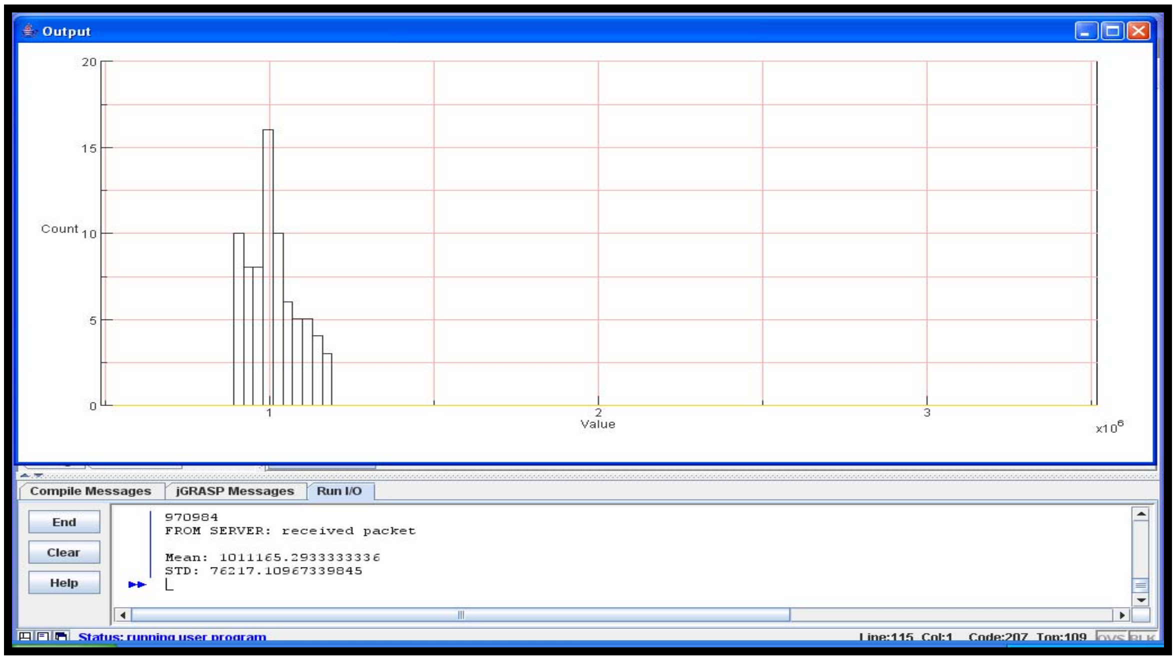
```
1573690
FROM SERVER: received packet

Mean: 1024221.2307692305
STD: 78634.92291313874
```

Figure 15. Sample run number two at 158 meters with filtering

Because radio waves travel at 3 x 10^8 m/s, the theoretical round trip time for 158 meters is 1.1 µs. However when the Java application was run 30 times, a mean of 1.0 ms was obtained, yielding a difference in magnitude of over 1,000 times that of the theoretical propagation time. A number of reasons can be attributed to this large difference; including, the large processing time that it takes for the wireless card to modulate the signal, the time it takes the OS to service other processes taking place before it services the networking data packets, and the large processing times attributed to the operations of the Java Virtual Machine (JVM).

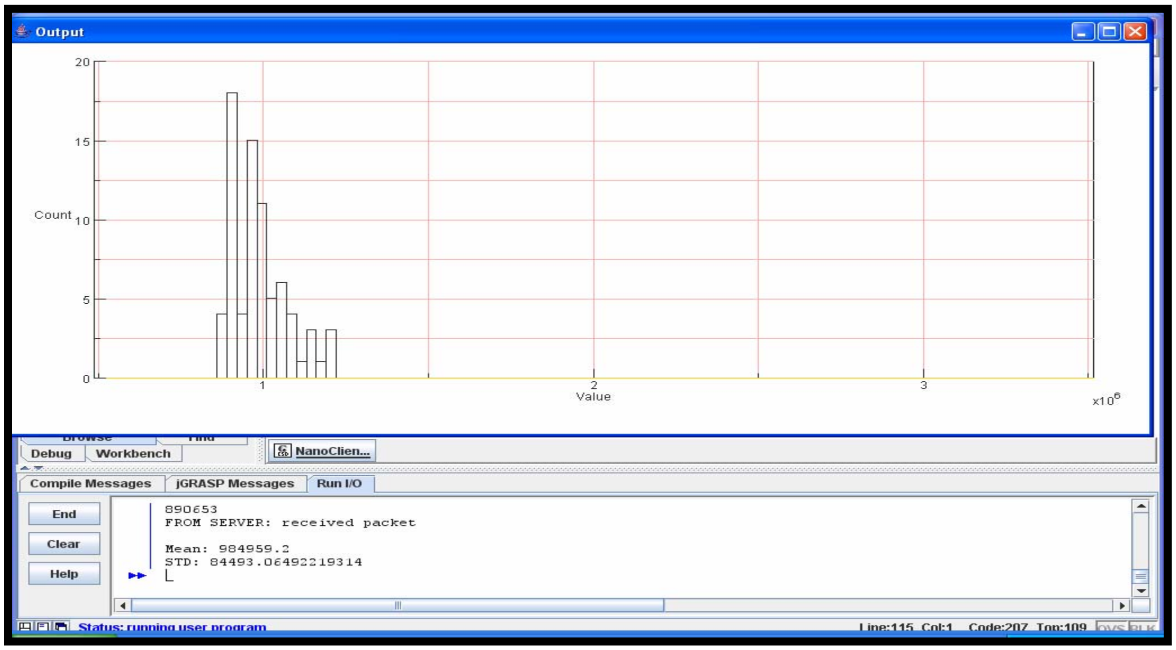
Measurements were then performed at 83 meters, Figures 16-17, with the intention of determining whether it is possible to resolve changes in distance using this methodology. Since the distance between the client and server is halved, a corresponding decrease in RTT is anticipated.



```
970984
FROM SERVER: received packet

Mean: 1011165.2933333336
STD: 76217.10967339845
```

Figure 16. Sample run number one at 83 meters with filtering



```
890653
FROM SERVER: received packet

Mean: 984959.2
STD: 84493.06492219314
```

Figure 17. Sample run number two at 83 meters with filtering

The results at 83 meters however suggest otherwise. From the histograms there is very little difference in RTTs from times obtained at 158 meters, with the average RTT at 83 meters resting at 1.0 ms. The results however are aligned with the conclusion that signal propagation times are easily overwhelmed by large processing times. A decrease in 83 meters corresponds to a decrease in 0.5 μs RTT, which goes undetected because measured RTTs are 1,000 times larger.

IV. DETERMINING DISTANCES USING BEACON ROUND TRIP TIMES

A. INTRODUCTION

An 802.11 setup used to determine distances using layer four TCP packets and IP-based addressing is not practicable to measure distances of any value because of the limitations imposed by markedly high processing times. Rather, in order to trim down the time it takes to process TCP/IP packets, it is necessary to capture link layer beacon frame propagation times instead.

B. PACKET ENCAPSULATION

Figure 19 below illustrates the simple encapsulation process for a chunk of data that initiates at the application layer.

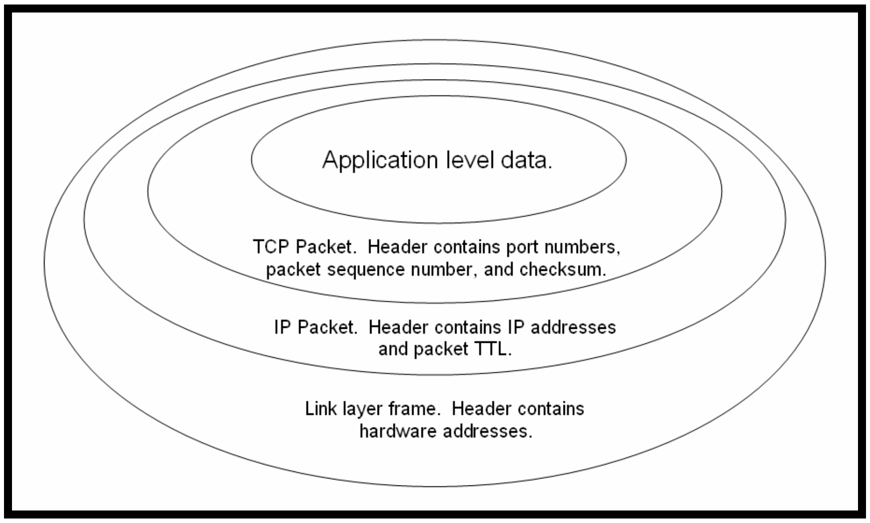

Application level data.
TCP Packet. Header contains port numbers, packet sequence number, and checksum.
IP Packet. Header contains IP addresses and packet TTL.
Link layer frame. Header contains hardware addresses.

Figure 18. Data Encapsulation

Normally data is created at the top of the protocol stack, or the application layer. The application data is then sent down the stack to be processed by encapsulating the data with additional protocol information, or headers. Each layer possesses different header information relevant to the functionality of that particular layer. As an example, the network layer uses source and destination IP addresses as part of its header information. This information is worthless to the other protocol layers, so the encapsulation process effectively isolates or obscures the header data until the packet arrives at a layer that can use the information, for the case of layer three, the networking layer uses this information to route packets from a network source node to a destination node. The higher level information is stripped to expose pertinent data. Whatever remains underneath is used by the protocols to move or process the data to the next layer. Packets that make numerous trips through routers go through a repeated process of stripping and encapsulation, which significantly adds to the system processing time. At the core of the encapsulated packet is the data, and not until all header layers have been stripped that the data is available to the destination application.

C. THE BEACON FRAME

Beacons are layer two frames that are important for a number of reasons. During passive searches, a node that wishes to associate to a particular BSS uses the beacon to determine the identity of the access point through the SSID.

Additionally, the beacons are important because they contain bits that help manage power saving functions. Because wireless nodes are often running on a battery source, it is essential that scarce power resources as efficiently handled. 802.11 sleep mode allows a node that is sitting idle to power down until there is a resumption of activity. During a period of inactivity, sleep mode ensures that nodes do not lose important data. The access point is responsible for sleep mode because it maintains a record of which nodes have sleep mode enabled. For these nodes, the access point buffers data packets until a polling request is initiated by the node. The beacon pulse may contain data informing a node that there is currently frames buffered awaiting delivery and that a station is requested to wake-up.

Lastly, the beacon frames allow all associated nodes to maintain synchronization to the access point, because each beacon pulse, which normally is transmitted every 100 ms, contains a timestamp of exactly when the beacon frame is sent from the access point. Nodes use the timestamp to update and synchronize their clocks to match the access point's clock for the purpose of network management. The timestamp therefore becomes an important parameter that is to be used in calculating distances. Figure 20 below is an illustration of an 802.11 beacon frame.

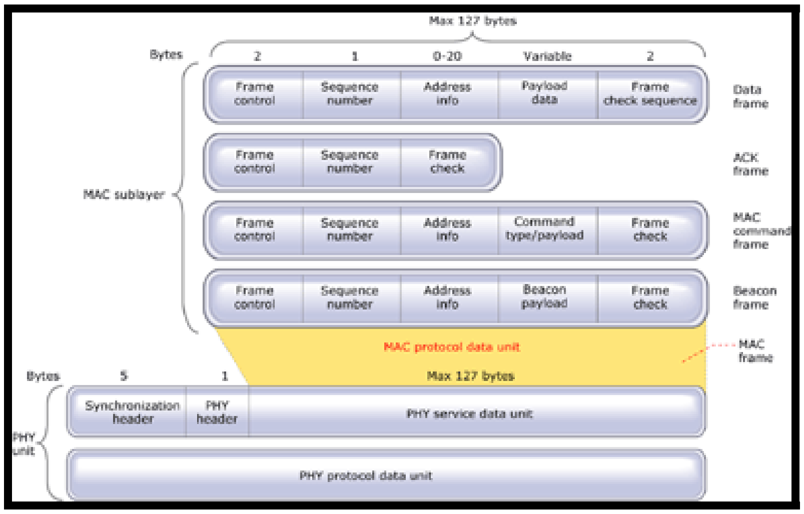


Figure 19. Beacon frame embedded in MAC layer

D. LINUX ACCESS POINT

There is a large variety of commercially available 802.11 access points. However many of these access points are operated by proprietary source code embedded in firmware that is not readily accessible to the public. It is possible, however, to use Linux as a platform that operates as an access point. Because Linux is open source, there is the flexibility of tailoring the access point configurations to particular needs, such as establishing a firewall, customizing network routing, or instituting NAT. For the purpose of this thesis, using Linux makes it much easier to get to the source code responsible for extracting link layer beacon information. One of the purposes of this project, then, involves configuring Linux and modifying its available support applications to establish an access point in order to capture beacon transmission times. Fedora Core version 4, was chosen as the Linux platform that is most suited for this project because of the availability of its support applications.

E. NETGEAR NETWORK INTERFACE CARD & THE PRISM 54 PROJECT

The Netgear WG511 network interface card, Figure 21, was chosen for this project because it contains the Intersil Prism GT Duette ISL3890 chipset. Unlike many other available chipsets on the market, such as certain models of the Atheros and the Intel based chipsets, it is possible to configure the Intersil chipset to work as an infrastructure access point. This is because the chipset contains firmware that is responsible for generating access point beacons.

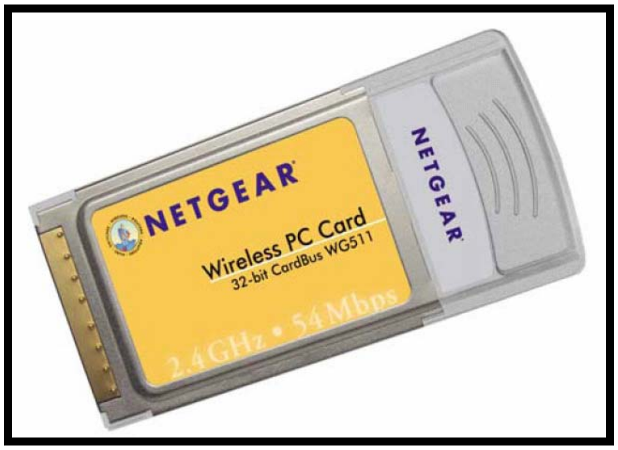


Figure 20. Netgear WG511

A driver is necessary to get the Netgear card to communicate with the Linux operating system. The Prism 54 driver was chosen because it provides compatibility for the Netgear device and the Fedora Core 4 environment. Additionally, the driver contains support for access point functionalities, namely it manages 802.11 frames that are passed to it from the network device.

F. LINUX ACCESS POINT SETUP

Although the project specifically involved three main elements, a Netgear WG511 card, the Prism54 driver and a Linux Core 4 OS, the subsequent instructions can be used in general to setup a customizable access point using any other compatible components. For example, it may be possible to us an Intersil based chipset with a HostAP driver running on a Linux Debian distribution.

The setup involved a lot of trial and error because there is much variation in how a particular release or distribution of Linux is configured, making this the most difficult part of the project. The location of the particular configuration file must be found, the parameters modified, and the kernel reset to test if the changes have the correct intended effect. Given that kernel source code is about 32 MB in size, compiling it may take as much as a couple of hours because process modules have to be built and linked. Sources on the Internet may be useful but be aware that instructions may be tailored for a particular hardware configuration, which may not be compatible with the Intersil, Prism54, Fedora Core 4 configuration. The following is an overview of the setup:

1. Determine hardware compatibility requirements for the Netgear WG511, 802.11 b/g wireless devices.
2. Install Linux Fedora Core 4 with native kernel 2.6.11-1.1369.
3. Upgrade to Linux kernel 2.6.18 (optional).
4. Reconfigure 2.6.18 kernel source tree with Full MAC Prism54 support using X-configuration utility.
5. Compile Subversion repository bleeding edge version of the Prism54 driver using gcc and bind to 2.6.18 kernel (optional with 2.6.18).
6. Compile hostap driver using gcc and bind to kernel (optional).
7. Install driver firmware add-on support, version 1.0.4.3, for 802.11b/g Intersil Prism GT-Duette ISL 3890 chipset.
8. Configure kernel for TCP/IP wireless interface (optional).
9. Configure 802.11 settings for wireless interface (optional).
10. Configure access point to forward packets and route traffic (optional).

11. Configure access point as a DHCP server (optional).
12. Configure access point to forward DNS requests (optional).

G. THE NETWORK DRIVER

The network driver acts as interface between the Linux operating system (OS) kernel and the hardware network interface card (NIC). When a packet is sent from the kernel to the hardware, the network driver is responsible for the commands that toggles the correct hardware registers to encapsulate the packet. Additionally it manages the radio modulation and signaling of the packet.

In general, the native kernel cannot feasibly contain enough control commands to account for every possible specific detail, of every NIC that is available on the market. The driver is designed to be portable and can be loaded into the kernel when a particular hardware is chosen. For the more popular models of hardware, various versions of the Linux kernel may already have a loadable kernel module driver as part of the kernel distribution package, and all that is needed is to bind the driver to the native kernel source tree.

The availability of the driver will determine whether the NIC can be used with Linux because the driver is required for the kernel and the hardware to communicate. A common mistake is to assume that a Linux driver for any NIC will always be available.

H. NETWORK INTERFACE CARD CHIPSET

To determine the availability of the driver, the identity of the hardware must be known. Compatibility with a particular driver is a function of the NIC chipset manufacturer because each manufacturer has its own proprietary design. In general, support for Linux wireless networking is limited. So often time open source groups have had to reverse engineer the design of device drivers in order to make a particular wireless device compatible with Linux. For example, Netgear does not provide a Linux driver for its WG511 NIC, which is based on an Intersil Prism GT Duette ISL3890 chipset.

However, the Prism54 group (http://www.prism54.org) has developed an ISL3890 driver that when used with a firmware add-on allows devices with the Intersil Prism GT Duette ISL3890 chipset to operate in a Linux environment. To determine the chipset inside, the card identification is usually helpful. For example, the FCC wireless device search engine can be used to locate detailed information about the chipset through the FCC ID number located on the back of the device. The manufacturer of the card may have technicians who can verify the chipset ID. For PCI devices, the lcpci command will list all connected PCI devices, which may include the device identification. In general if the driver is already loaded in the kernel, the Network Configuration Tool will indicate the device identification. Lastly, doing a Google search for Netgear WG511 generally will reveal that it contains the ISL3890 chipset.

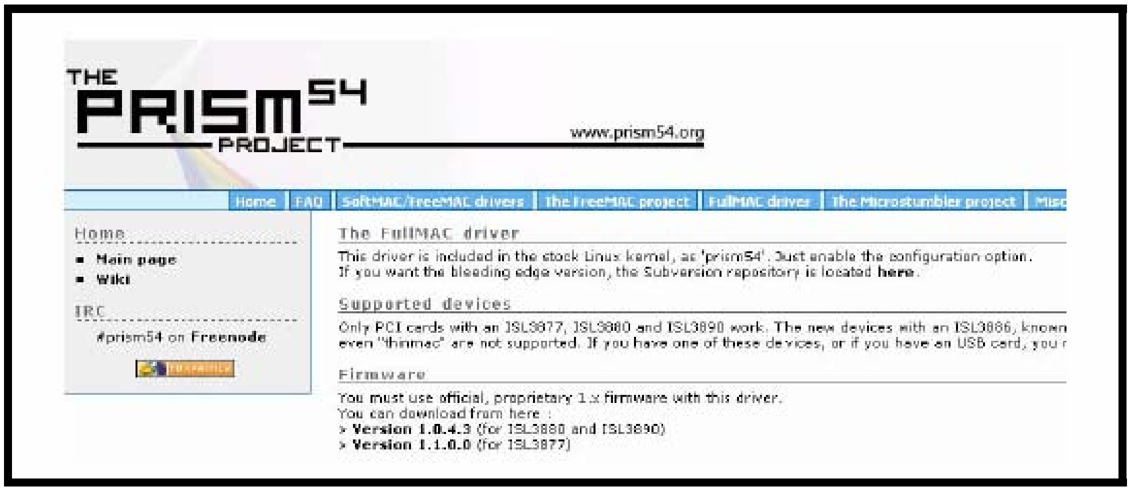
The chipset technology must also be known. For wireless LANs, hardware technologies are generally categorized into three groups: a) IEEE 802.11 legacy which include 802.11-FH (frequency hopping) and 802.11-DS (direct sequence spread spectrum), b) IEEE 802.11b, c) and IEEE 802.11a and 802.11g. Locating a suitable driver then depends on what technology the hardware is using. For example the Prism II chipsets use 802.11b technology and is compatible with the prism2 driver. The Intersil Prism GT Duette ISL3890 chipset however is based on 802.11 b/g technology and is compatible with the prism54 driver. Because each chipsets use different 802.11 technologies the drivers for each chipsets are not interchangeable.

The Linux driver will depend on what NIC interface is used. ISA, PCI, PCMCIA, Cardbus, PLX, USB interfaces, etc. in general will require different drivers. This is because the each hardware interface type requires different software instructions related to the transfer of data. For example there are four different versions of the prism2 driver: prism2_cs.o for PCMCIA, prism2_plx.o for PLX, prism2_pci.o for PCI, and prism2_usb.o for USB.

I. FIRMWARE

The NIC firmware is software that is embedded in the device that includes control instructions for the device. It is proprietary microcode that is unique to the device and is stored in read only memory on a semi-permanent storage device. The microcode is nonvolatile and may be updated depending on the design of the device. It is more common to find NICs on the market that have firmware that is designed exclusively for Windows based platforms, therefore without a driver that contains an interface to firmware instructions, compatibility between the NIC and Linux is not possible. Some chipset manufacturers are unwilling to release the microcode source to open source programmers, and the microcode is often protected by copyrights. Therefore in order to attain compatibility between a Windows based NIC and a Linux computer, the options are to replace the Windows firmware with a Linux microcode, find a Linux driver that has a Windows firmware interface, or locate a firmware add-on that when used in conjunction with a Linux driver provides an interface to the Windows firmware.

The firmware add-on allows the device manufacturer to provide support to the Linux driver community without directly releasing copyrighted materials. The Linux driver community has access to the micro-code source and thus can more readily program Linux drivers that can interface with the micro-code. The firmware add-on is not firmware but a micro-code extension, meaning the code is loaded every time Linux is booted and when the device is present. For the Intersil Prism GT Duette ISL3890 chipset, the latest firmware add-on release is 1.0.4.3.arm and is obtained from the Prism54 project, Figure 22. The firmware add-on is then generally installed in the /lib/firmware directory with a path to it annotated in the /etc/hotplug/firmware.agent file with FIRMWARE_DIR = /lib/firmware.


THE PRISM54 PROJECT
www.prism54.org
Home | FAQ | SoftMAC/FreeMAC drivers | The FreeMAC project | FullMAC driver | The Microstumbler project | Misc
Home
Main page
Wiki
IRC
#prism54 on Freenode

The FullMAC driver

This driver is included in the stock Linux kernel, as 'prism54'. Just enable the configuration option.
If you want the bleeding edge version, the Subversion repository is located **here**.

Supported devices

Only PCI cards with an ISL3877, ISL3880 and ISL3890 work. The new devices with an ISL3886, known
even "thinmac" are not supported. If you have one of these devices, or if you have an USB card, you r

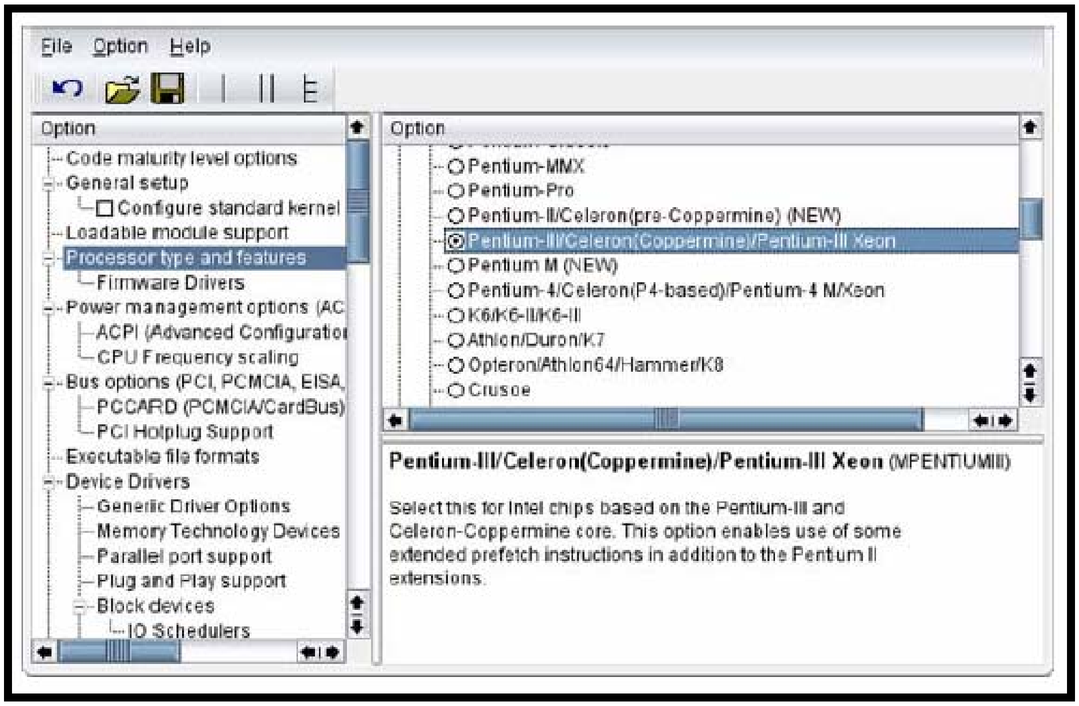
Firmware

You must use official, proprietary 1.x firmware with this driver.
You can download from here :
> **Version 1.0.4.3** (for ISL3880 and ISL3890)
> **Version 1.1.0.0** (for ISL3877)

Figure 21. Intersil Prism GT Duette ISL3890 Chipset Firmware Add-On

J. LINUX LOADABLE KERNEL DRIVERS AND COMPILING THE KERNEL

The Linux kernel may already have the driver module sources as part of the base kernel package. Once the correct driver is determined for the chipset, check the kernel configuration files to see if the driver sources are available. If they are present, configuration involves enabling the driver that is needed and recompiling the kernel. The make xconfig utility, Figure 23, is a GUI based configuration utility that provides the capability to pick and choose the modules that are needed.



○ Pentium-MMX
○ Pentium-Pro
○ Pentium-II/Celeron(pre-Coppermine) (NEW)
◉ Pentium-III/Celeron(Coppermine)/Pentium-III Xeon
○ Pentium M (NEW)
○ Pentium-4/Celeron(P4-based)/Pentium-4 M/Xeon
○ K6/K6-II/K6-III
○ Athlon/Duron/K7
○ Opteron/Athlon64/Hammer/K8
○ Crusoe

Pentium-III/Celeron(Coppermine)/Pentium-III Xeon (MPENTIUMIII)

Select this for Intel chips based on the Pentium-III and Celeron-Coppermine core. This option enables use of some extended prefetch instructions in addition to the Pentium II extensions.

Figure 22. make xconfig Configuration Utility

During the configuration, an option of choosing whether the module is a loadable kernel module or a module that is permanently bound to the kernel is provided. A ‘Y’ response indicates that the module is to be bound to the kernel, an ‘M’ response indicates that the module is to be a loadable kernel module and an ‘N’ response indicates that the module is not desired. Once the desired configuration is chosen, compile the base kernel using make bzImage. make bzImage is a utility that compiles and creates a compressed binary image of the base kernel which is executed when Linux boots. After the kernel image is created the loadable kernel modules must be built with the make modules utility, which is used to compile the driver module source codes. What results from this procedure is a large number of compiled object files with .o extensions that were chosen during the configuration step above. These files need to be moved to the directory where the kernel will be able to locate them, which normally is the /lib/modules directory. Use the make modules_install utility to move these object files to this directory.

K. EXTERNAL DRIVER MODULES

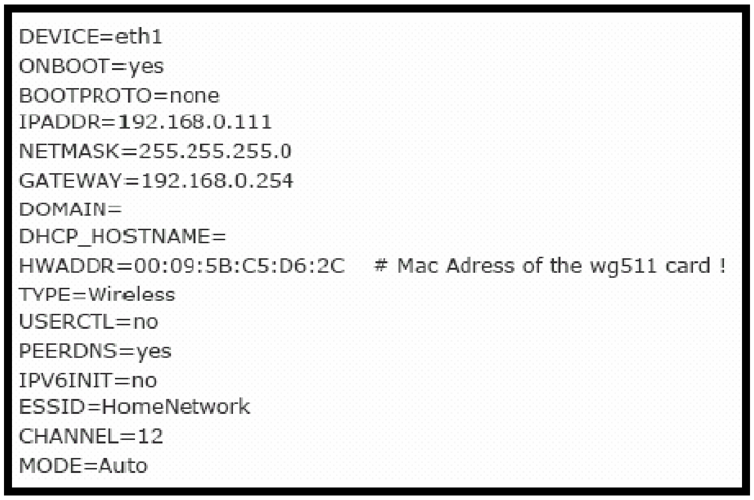
If the driver module is not part of the kernel package, the Linux kernel needs to be modified by adding the source code of the driver to the Linux kernel source tree and compiling the kernel using the procedures above. The driver source code generally contains a Makefile, which is basically a configuration file that is run by the make utility. The make utility uses the Makefile to locate the driver and kernel source files, to compile and link the code and to create an executable binary that the OS can use. The KERNEL_PATH parameter within the Makefile may have to be modified, in order for the make utility to locate the correct path to the kernel source files. As above the make utility creates an object file which is installed into the appropriate directory using the make install utility.

L. LOADING THE MODULES

If the modules were configured as loadable modules, they can be loaded using the modprobe and insmod utilities. The Linux kernel modules can be explicitly loaded using the rmmod command. A loaded module can be thought of as being a complete part of the base kernel and its rights and privileges to OS resources are managed directly by the base kernel. This means if the modules are poorly coded, a loaded module may cause the kernel to crash similar to any other statically or demand loaded kernel modules. The module object files are re-locatable, meaning the files are not linked to run from a permanently assigned address. This makes it possible for the kernel to link to the object files on demand, using the modprobe and insmod utilities.

M. WIRELESS CONFIGURATIONS

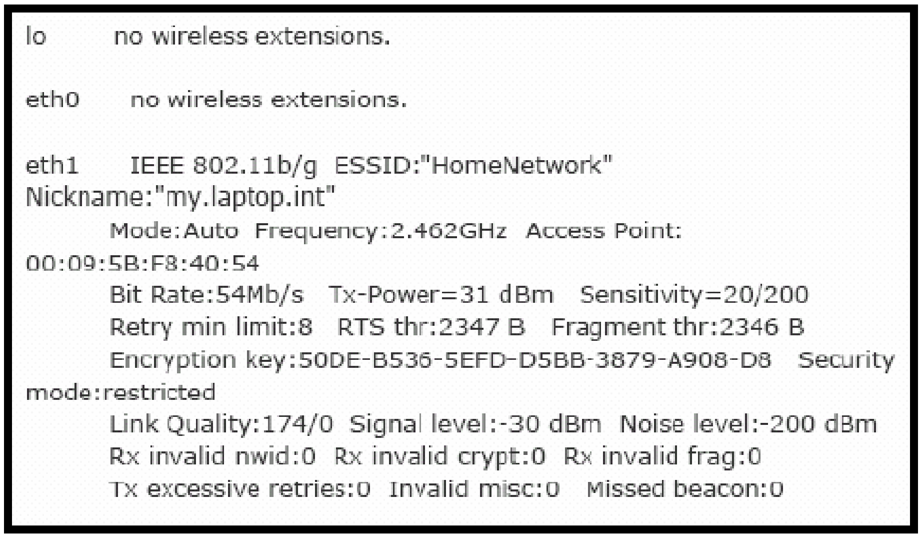
Configuration of the wireless device parameters can be done by modifying script files directly or using the iwconfig utility, Figure 24. The script files are generally located in /etc/sysconfig/network-scripts directory. WEP keys are stored in the keys file under the same directory path.



```
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=none
IPADDR=192.168.0.111
NETMASK=255.255.255.0
GATEWAY=192.168.0.254
DOMAIN=
DHCP_HOSTNAME=
HWADDR=00:09:5B:C5:D6:2C    # Mac Adress of the wg511 card !
TYPE=Wireless
USERCTL=no
PEERDNS=yes
IPV6INIT=no
ESSID=HomeNetwork
CHANNEL=12
MODE=Auto
```

Figure 23. ifconfig Script File

Alternatively, the iwconfig utility, Figure 25, can be used with the switches essid, mode, and key to set the basic parameters that are needed to establish a wireless connection.



```
lo        no wireless extensions.

eth0      no wireless extensions.

eth1      IEEE 802.11b/g  ESSID:"HomeNetwork"
Nickname:"my.laptop.int"
          Mode:Auto  Frequency:2.462GHz   Access Point:
00:09:5B:F8:40:54
          Bit Rate:54Mb/s   Tx-Power=31 dBm   Sensitivity=20/200
          Retry min limit:8   RTS thr:2347 B   Fragment thr:2346 B
          Encryption key:50DE-B536-5EFD-D5BB-3879-A908-D8   Security
mode:restricted
          Link Quality:174/0  Signal level:-30 dBm  Noise level:-200 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Figure 24. iwconfig Utility

N. NETWORK CONFIGURATIONS

Network parameters are configured using the script files to change static IP addresses, the network mask, and the gateway and domain names. To establish DHCP services use the dhclient utility. IP routes can also be established if the Linux computer is to be used as a router. If the computer is to be used as an access point the NIC must have a chipset that is bridging capable. For example, the iwconfig eth1 mode Master option allows the computer to serve as an access point, using eth1 as the wireless interface.

O. PRISM 54 BEACON ROUTINE

The Prism54 driver contains the instructions that control the sending of network beacons. The complete source code for the bleeding edge driver is available on the Subversion directory website (http://svnweb.tuxfamily.org/). The source code is divided into a number of different functions:

1. pci_dev: Device Interrupt Handler, Network Interface Control & Statistical functions, and Network device configuration functions.
2. pci_eth: Network Interface functions.
3. pci_hotplug: Module initialization functions.
4. pci_38xx: Device Interface & Control functions.
5. ap: PIMFOR netdev interface (Proprietary Intersil Mechanism For Object Relay). PIMFOR is a simple request-response protocol used to query and set items of management information.
6. ioctl: allows the code to manage communications with the device driver outside the typical read/write operations of data.
7. wds: Wireless Distribution System allows for the establishment of interconnected access points through wireless connections. It is an IEEE 802.11 protocol that allows for expansion of a BSS without relying on a wired backbone.

8. oid_mgt: Converts between channel and freq.

The code of interest, however, is located in the ioctl function, which is responsible for variety of device-specific control purposes, including the beacon management frames. The following piece of code is responsible for the beacon or probe response payload header, depending on whether the station is serving as an access point or as a wireless client. The probe response is a reply from the wireless client that includes the station's specific parameters and its supported data rates.

```
/* Beacon/ProbeResp payload header */
struct ieee80211_beacon_phdr {
        u8 timestamp[8];
        u16 beacon_int;
        u16 capab_info;
} __attribute__ ((packed));
```

In the event the station is a client, received beacons are processed to extract BSS information belonging to the access point or peer station operating in ad-hoc mode:

```
static void
prism54_process_bss_data(islpci_private *priv, u32 oid, u8 *addr,
                        u8 *payload, size_t len) {
        struct ieee80211_beacon_phdr *hdr;
        u8 *pos, *end;
        if (!priv->wpa)
                return;
        hdr = (struct ieee80211_beacon_phdr *) payload;
        pos = (u8 *) (hdr + 1);
        end = payload + len;
        while (pos < end) {
                if (pos + 2 + pos[1] > end) {
```

```
            printk(KERN_DEBUG "Parsing Beacon/ProbeResp failed"
                  "for " MACSTR "\n", MAC2STR(addr));
            return;     }
        if (pos[0] == WLAN_EID_GENERIC && pos[1] >= 4 &&
           memcmp(pos + 2, wpa_oid, 4) == 0) {
            prism54_wpa_ie_add(priv, addr, pos, pos[1] + 2);
            return;    }
        pos += 2 + pos[1];    }   }
```

In order to configure the wireless station to work as an access point it is necessary to use the wireless utilities to establish Master mode. The following piece of code switches the wireless device into Master mode:

```
prism54_set_mode (struct net_device *ndev, struct iw_request_info *info,
          __u32 * uwrq, char *extra) {
    islpci_private *priv = netdev_priv(ndev);
    u32 mlmeautolevel = CARD_DEFAULT_MLME_MODE;
    if ((*uwrq == IW_MODE_MASTER) && (priv->acl.policy !=
    MAC_POLICY_OPEN))
        mlmeautolevel = DOT11_MLME_INTERMEDIATE;
```

P. CALCLULATING DISTANCES USING LAYER TWO FRAMES

The stats_timestamp parameter may be the key to being able to effectively calculate distances using layer two frames:

```
struct iw_statistics *
prism54_get_wireless_stats(struct net_device *ndev) {
    islpci_private *priv = netdev_priv(ndev);
    /* If the stats are being updated return old data */
    if (down_trylock(&priv->stats_sem) == 0) {
        memcpy(&priv->iwstatistics, &priv->local_iwstatistics,
            sizeof (struct iw_statistics));
        /* They won't be marked updated for the next time */
        priv->local_iwstatistics.qual.updated = 0;
        up(&priv->stats_sem);
    } else
        priv->iwstatistics.qual.updated = 0;
    /* Update our wireless stats, but do not schedule to often
     * (max 1 HZ) */
    if ((priv->stats_timestamp == 0) ||
      time_after(jiffies, priv->stats_timestamp + 1 * HZ)) {
        schedule_work(&priv->stats_work);
        priv->stats_timestamp = jiffies;        }
    return &priv->iwstatistics; }
```

Modification of the wireless client driver and the access point driver is necessary in order to properly capture correct round trip frame times. A possible modification would involve having the wireless client automatically send back a probe response whenever it receives a beacon from the access point. The access point having kept track of its initial beacon timestamp calculates the difference in time between the initial timestamp and when it receives the return probe response from the client. The difference in time would then be correlated to:

$$RTT = Total\ Time = Proc\ 1 + Proc\ 2 + Prop\ 1 + Prop\ 2$$

Because the beacons and probe responses are being processed as layer two frames, it is expected that total processing times (Proc 1 and Proc 2) will be substantially reduced.

V. SIGNAL STRENGTH

A. SIGNAL STRENGTH UNITS OF MEASURE

802.11 signal strengths are often reported in units of milliwatts (mW), db-milliwatts (dBm), Received Signal Strength Indicator (RSSI), or percentage values. Once one value is known, the other values can be calculated through simple conversion methods. For example, dBm can be converted from mW through the following relation:

$$dBm = 10 * \log (mW)$$

RSSI values are arbitrarily assigned by the device manufacturer and are based on the device's most practical range of power. RSSI values range from 0-255 or the range represented by 1 byte, but manufacturers can subjectively choose RSSI ranges anywhere between 0 and 255. For example, CISCO chooses to correlate RSSI 0, the device's reception threshold sensitivity, to -96dBm and RSSI 100, the device's highest signal strength value, to -20 dBm, even though signal strengths may extend up to or beyond 20 dBm. RSSI values are chosen because wireless devices are not truly capable of measuring 0% signal strengths. This is because it is incapable of telling the difference between an environment with completely no signal and an environment with a signal but no packets being transmitted. Additionally, since a typical wireless access point power drops below -20 dBm at 5 feet, it is practical and reasonable to assign 100% signal strength to a measured power of -20 dBm.

Percentage values are assigned relative to RSSI values. For example, Netstumbler signal strength meters have an RSSI range of 0-50. RSSI 0 therefore corresponds to 0%, RSSI 25 corresponds to 50% and RSSI 50 corresponds to 100% signal strengths.

Figure 25 is a graphical representation of the relationship between the various units of measure for signal strength.

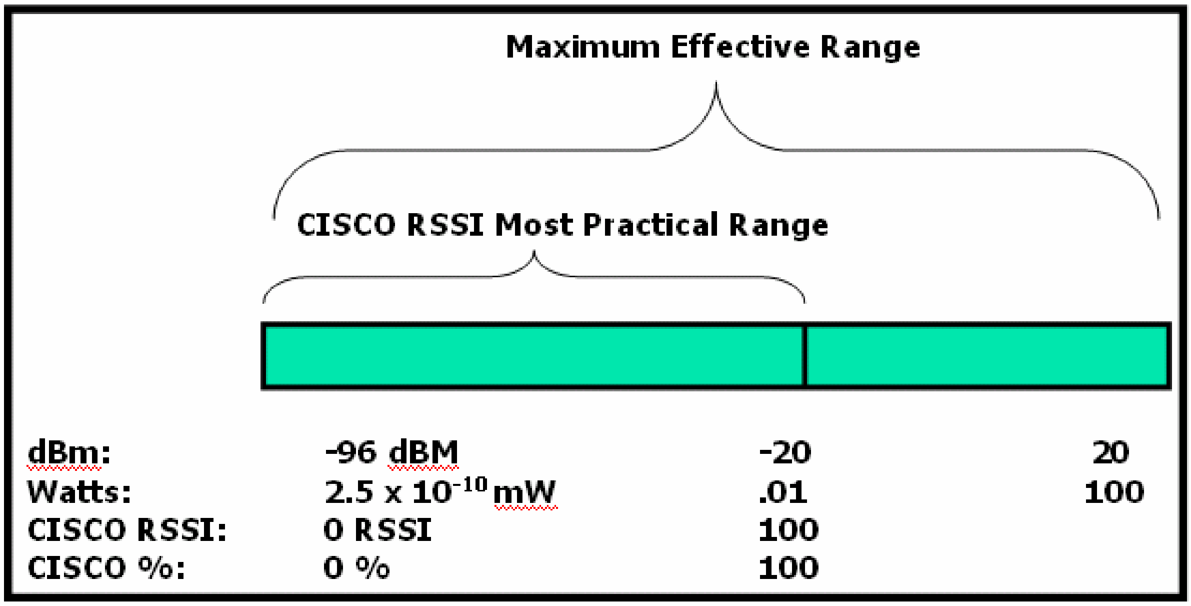


Figure 25. Signal Strength Units of Measure

RSSI correlates distance based on received power levels. Since the wireless nodes already have radios for communication, RSSI is a low-power method of measuring distance at almost no computational cost. Thus, measuring distance through RSS is highly desirable as the existing hardware can be used, mobile nodes can perform the calculations independently, and deployment is simple.

B. "INVERSE-SQUARE" RELATION

Power loss over the air, given constant atmospheric conditions, follows the inverse square law:

$$Power \ \alpha \ 1 / distance^2$$

Receiver threshold sensitivity is approximately 2.5 x 10^{-10} mW for a typical 802.11 wireless card. If a transmitter is located at the center, the surface area of the sphere can be used to approximate power at a given distance from the transmitter. This is because power is omni-directional and is therefore proportional to the surface area of a sphere, $4\pi r^2$. Distance from the transmitter is then the radius of the sphere. The inverse relation is due to the loss of power as you move further from the transmitter. Because of this relationship, the following signal strength graph is expected, Figure 26:

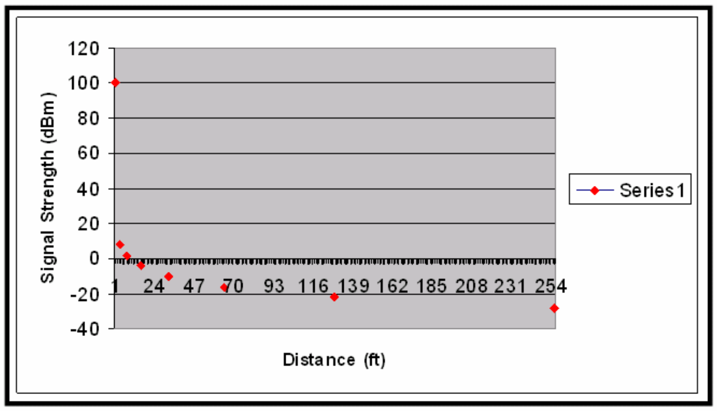


Figure 26. Signal Strength (dBm) vs Distance (ft)

Once the data is extrapolated, determination of distance can be achieved by cross referencing signal strength values from the graph.

C. METHODOLOGY

RSSI values are used to determine a mobile node's distance from an access point. Because of the inverse relationship between power and distance, it is possible to produce a signal strength map through systematic sampling of the environment. The first step in RSSI-based localization is estimating the distance between two nodes, given the signal strength received by one wireless node from the other. A signal strength-distance correlation graph is created using various locations where signal strengths are measured and recorded. The correlation map is extrapolated from a finite set of collected measurements and is used as a reference to predict actual distance values. Because signal strengths fluctuate, sometimes with large deviations due to ambient conditions and noise, it is important to ensure that environmentals are the same when correlating distances to signal strengths. For example, with Linux-based systems, the iwconfig utility includes functionality to measure noise which can be used to eliminate signal strength measurements taken under high noise conditions. An algorithm can be used to automate this process. Signal strength values that have a noise value above a certain threshold are removed, while those values obtained under lower noise conditions are given higher weightings in the calculations process. Once a signal strength mapping is determined, signal strengths are cross-referenced to the correlation map to calculate distance. If the signal strength measures exhibit good precision and accuracy, it may be possible to determine positions using distances taken from the signal strength map.

The test bed is comprised of one Dell Celeron mobile laptop configured with Linux Fedora Core 4 running as an access point using a Netgear WG511 wireless card rated at 31 dBm maximum effective transmit power. An additional Dell Celeron wireless client, configured with Linux Fedora Core 4, was used running the native supplicant. The access point was placed at home base on a softball field. Using the Linux client and the iwconfig utility, signal strength values were captured at 25 ft intervals from the access point, Figure 27. Thirty signal strength measurements were taken at each location and averaged. These measurements were then plotted against distance from the access point and best fit line extrapolated through the data points.

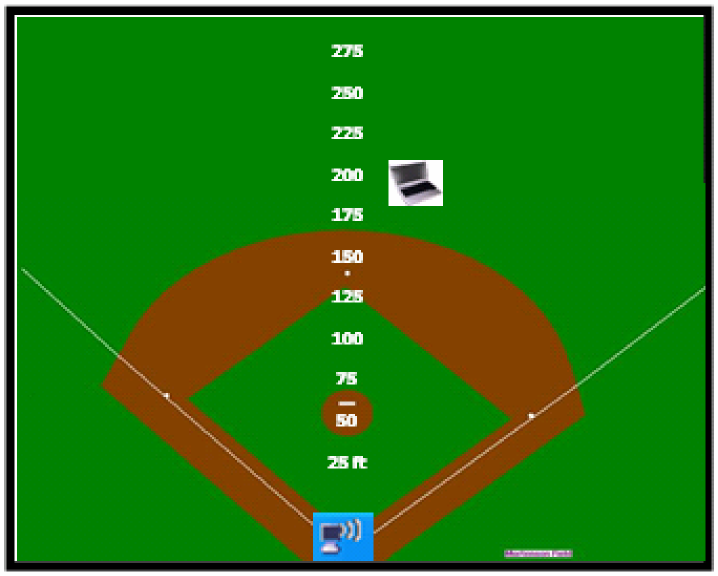
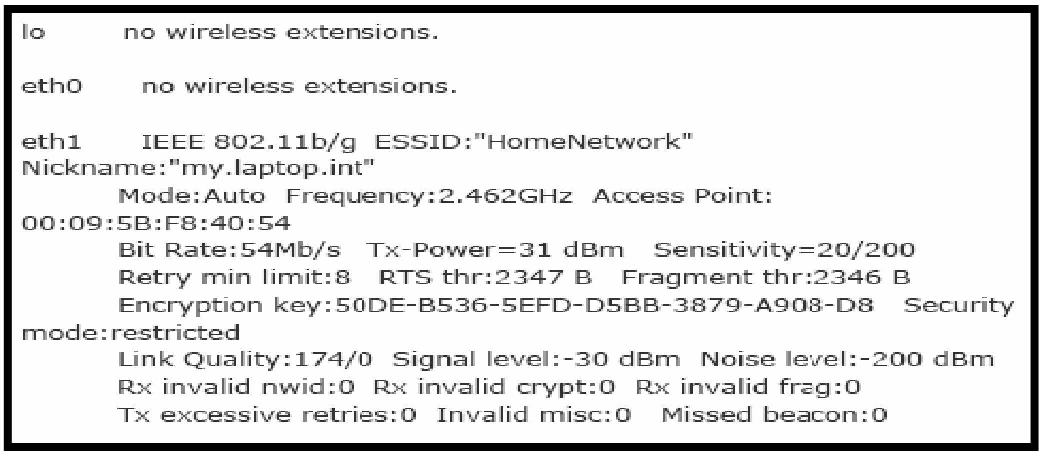


Figure 27. Signal Strength Measured at 25 ft Intervals from Access Point

A typical 802.11 wireless device has a rated output of 100 mW. Although not perfectly accurate because of power losses through the transmitter device, it can be assumed for simplicity that you see 100 mW of power at the tip of the device antenna.

Eleven positions were selected to perform range estimations. For each position, signal strength measurements were taken. The iwconfig tool was used to extract RSSI values, Figure 28. Measurements of RSSI values were then imported into Excel for statistical analysis.



```
lo        no wireless extensions.

eth0      no wireless extensions.

eth1      IEEE 802.11b/g  ESSID:"HomeNetwork"
Nickname:"my.laptop.int"
          Mode:Auto  Frequency:2.462GHz  Access Point:
00:09:5B:F8:40:54
          Bit Rate:54Mb/s   Tx-Power=31 dBm   Sensitivity=20/200
          Retry min limit:8   RTS thr:2347 B   Fragment thr:2346 B
          Encryption key:50DE-B536-5EFD-D5BB-3879-A908-D8   Security
mode:restricted
          Link Quality:174/0  Signal level:-30 dBm  Noise level:-200 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Figure 28. Linux's iwconfig

A second set of measurements was taken from the access point to 10 meters. Because the inverse-square relation states that RSSI values exponentially decreases with increasing distance, it was necessary to capture signal strength values close to the access point. In doing so, distance resolution is more accurate and precise because, near the access point there is a large change in RSSI for every given change in distance. Again, both laptops were configured to run in ad-hoc mode. With iwconfig running, received signal strengths were measured in increments of 1 meter.

Once the grid map is finalized, signal strength is equated to distance. To automate signal strength measurements, a bash script, iwDistance.sh, was implemented using Perl.

```
while [ 1 = 1 ]
        do  sleep 1  iwconfig 2> /dev/null | perl iwParse.pl
done
```

This shell script executes iwconfig every second. The output is blanked and piped to iwParse.pl. iwParse.pl is designed to parse iwDistance.sh's output and extract the dBm value. For iwParse.pl to work properly, the upper and lower limits of the RSSI of a particular location have to be determined.

```
while (<STDIN>) {
($dBmValue) = ($_  =~ /Tx-Power=(.*?) dBm/);
        if ($dBmValue >= 30 && $dBmValue < 40) {print "10 feet\n"}
        #more conditional statements }
```

D. RESULTS

In order to establish the reliability of received signal strength readings, a value of 95% was assigned to the confidence interval. By doing so, the upper and lower limits of the received signal strengths at a particular distance could then be determined. This value was chosen so that once the upper and lower limits were incorporated in iwParse.pl, its output will be more accurate as there will be a 0.95 probability that the confidence interval would contain the population mean.

RSSI was measured in increments of 25 feet and plotted versus distance, Figures 29 and 30.

Distance (ft)	average Signal Strength (dBm)	Power (mW)
25	-63	5.01E-07
50	-72	6.31E-08
75	-67	2.00E-07
100	-68	1.58E-07
125	-66	2.51E-07
150	-72	6.31E-08
175	-73	5.01E-08
200	-69	1.26E-07
225	-74	3.98E-08
250	-76	2.51E-08
275	-75	3.16E-08

Figure 29. Signal strength measurements

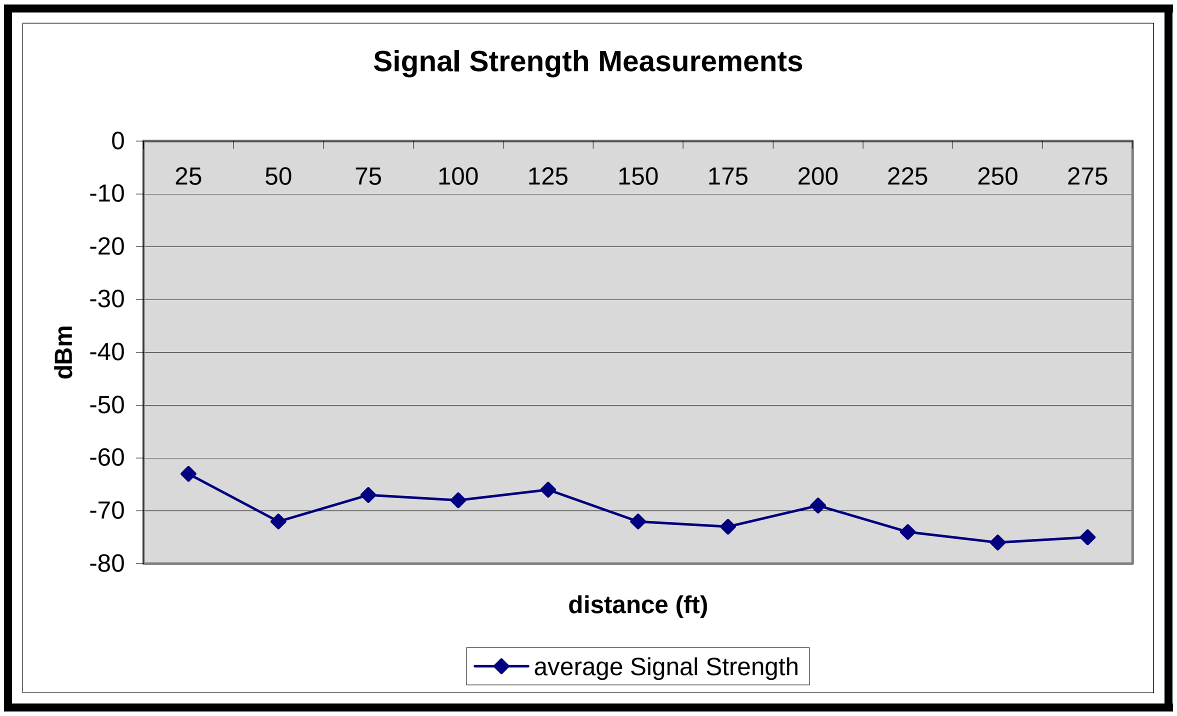


Figure 30. Signal strength versus distance, 25 to 275 ft.

From Figure 30, a decline in signal strength is noticeable. The apparent decrease in signal strength seems to follow the trend predicted by the inverse-square law. It was expected that a decrease in signal strength value will occur as the mobile client is moved away from the access point. However from the results it is not possible to accurately resolve distances using signal strengths taken at positions greater that 25 ft from the access point because RSSI changes so little for every given unit change in distance. Using the wireless devices in this setup, it is impossible to accurately resolve distances beyond 25 ft due to the limitations of the gear.

The following shows the results of signal strength measurements taken at distances less than 10 meters from the access point, Figure 31.

Distance (m)	Average Signal strength (dBm)	Power (mW)	Lower Limit	Upper Limit
1	-44.13333	3.86071E-05	-44.36	-43.91
2	-57	1.99526E-06	-57.36	-56.64
3	-69.4	1.14815E-07	-69.73	-69.07
4	-69.8	1.04713E-07	-70.16	-69.44
5	-66.1	2.45471E-07	-66.97	-65.23
6	-70.6	8.70964E-08	-71.68	-69.52
7	-76.53333	2.22161E-08	-77.53	-75.54
8	-78.3666	1.4566E-08	-78.81	-77.92
9	-76.13333	2.43594E-08	-76.62	-75.65
10	-74.36666	3.65876E-08	-74.70	-74.03

Figure 31. RSSI at distances less than 10 meters from access point

Within 1 to 3 meters, for every 1 meter change in distance there is a large change in RSSI, Figure 32. The change in RSSI values are large enough that for the wireless gear used in this setup, it is possible to accurately resolve distances. However, there are overlaps between the upper and lower limits of the received signal strength established by the 95% confidence interval for each pre-determined location. Thus, it is practically impossible for iwParse.pl to determine the wireless node's position once the node's distance from the access point exceeds 3 meters.

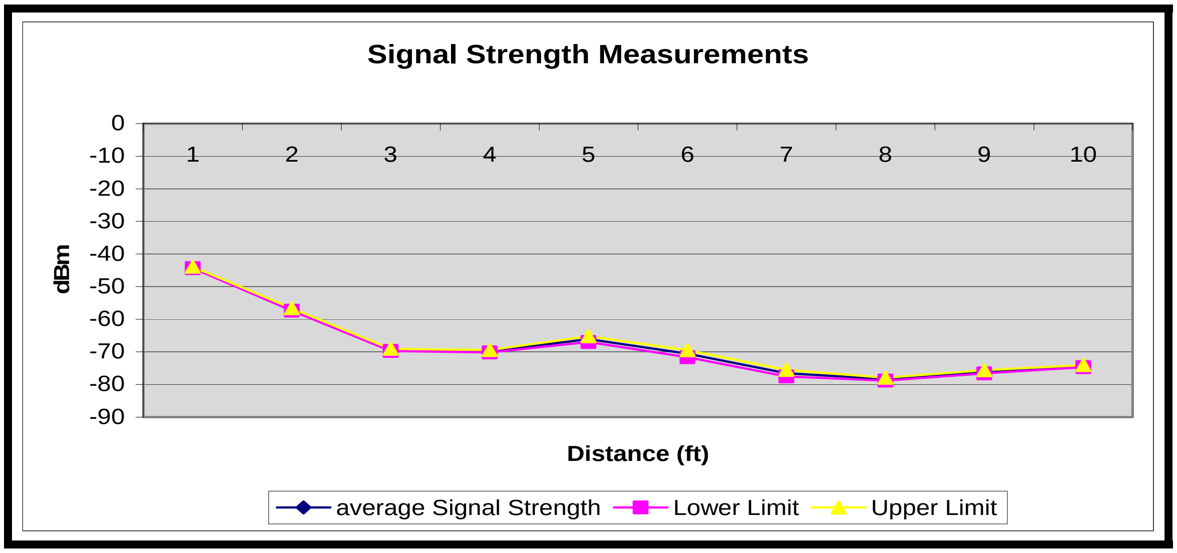


Figure 32. Signal strength versus distance, 1 to 10 meters

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

A. ROUND TRIP TIMES

The time it takes to send data over an 802.11 backbone can be used to determine positioning information between the clients and access points on the network. However, using round trip times to effectively determine distances highly depends on the processing time of the client-server pair and the propagation time of the radio signal over the air. For RTTs obtained through layer three packet transmissions, the processing times are too high to yield any information of value. Observed layer three total time, the packet processing and packet round trip propagation time, for a 200 meters setup is on the order of 1 ms with a standard deviation of 80 μs. The physical properties of radio signal transmissions over the air defines the speed of the radio signal as 3 x 10^8 m/s, or as a unit of data travels one meter in 3.33 ns. Therefore for every change in one meter of distance there is a corresponding change of 6.66 ns in round trip propagation time. For 200 meters, it takes a signal 1.33 μs to complete a round trip from a client to the server and back. Because the standard deviation is nearly 80 times greater than the round trip propagation time for a layer three packet to travel 200 meters, any changes in round trip time that arises due to a change in distance between the client and server, is easily overwhelmed by the deviations that result from conducting measurements.

Layer two frames may provide a solution to effectively using round trip propagation times as a way in obtaining distance. One possible method involves using 802.11 beacon frames. Because beacons are a link layer entity, much of the processing time it takes to package data is eliminated. When compared to data that originates from the application layer or any layer that is above layer two, there is a significant time premium that is saved by using layer two frames. Take for example a typical CPU operating at 1 GHz. A single instruction would at its best take 1 ns to complete. Assuming the processor is running a typical 50 processes, it would take 50 ns before the data unit can be serviced, if each process is a composed of a single instruction. As the data unit moves down the protocol stack, additional processing occurs. How much processing time is added to total time measurements is therefore a function of how many instructions there are, how many processes are running and how fast the processor is running. Additionally, if the data originates higher up in the protocol stack, processing time depends on what type of hardware is servicing the instruction at the particular layer. For example, the application layer is serviced by the main CPU, the network layer is service by the router CPU, and the link layer is serviced by the chipset embedded in the wireless card. By working with link layer frames, processing times from the main CPU and the router CPU is eliminated. Therefore, in order to be able to resolve changes in distance of one meter, the deviations in total processing time needs to be less than 6.66 ns, the time it takes a radio signal to complete a one meter round trip propagation path. Using link layer frames will lower total processing time measurements which will lead to lower processing time deviations, possibly achieving the nanosecond threshold. If this holds true, measuring distances using 802.11 signals will be possible.

B. SIGNAL STRENGTHS

Using signal strengths is an alternative means of wireless client positioning. The inverse square law defines the physical properties of wireless signal strengths and can be used to calculate distance through extrapolation of a signal strength versus distance graph.

Given the low power ratings for the typical wireless card and wireless access point, the data does not suggest that a robust method of client location is possible for the typical 802.11 wireless devices using signal strengths and the inverse-square power relation. However modifications to the wireless hardware may make positioning, throughout a larger distance range, possible, if the devices' transmit power can be increased such that there is a power distribution that extends further from the transmission source. The typical 802.11 wireless device transmits at a power level rated between 10 mW to 50 mW, or 10 dBm to 17 dBm, which corresponds to approximately 200 meters maximum effective range. Near the transmission source and within 5 meters, the signal strength power distribution drops off significantly as a function of distance as the client moves away from the source. At this range, there is an advantage for client location because for every small change in distance, there is a corresponding large change in signal strength, which can easily be detected. The problem, however, is that beyond 3 meters there is little resolution of distance because the inverse square curve levels off beginning at 3 meters. That is, for a significantly noticeable change in signal strength, there needs to be a large change in distance. Any small change in distance, such as one meter, which is a reasonable performance requirement, would go unnoticed.

Increasing the source hardware transmit power is a possible solution to generating an inverse square distribution that is more robust. If the "heel" of the curve, the area of the curve where power begins to level off, can be extended beyond 3 meters, client location using signal strengths may offer some significant advantages. Various wireless components are available with power ratings that are much higher than typical FCC compliant 802.11 hardware. For example, SMC Networks offers a wireless PC Card that transmits at a power rating that is twice the power rating of a standard commercially available wireless card. According to the company, the SMC 2532W wireless card transmits at 200 mW, or 23 dBm, which translates to a maximum effective range that extends beyond 800 meters. The SMC 2532W works with the Linux access point configuration because it contains the Intersil chipset, which is compatible with Fedora's Prism54 driver and firmware add-on.

Another possible solution involves using a directional antenna, Figure 33, to focus the wireless device's RF pattern so as to achieve higher antenna gain. Using directional antennas offer significant performance improvements, on the order of a two to eight times, corresponding to a 3dB to 9dB increase in gain.

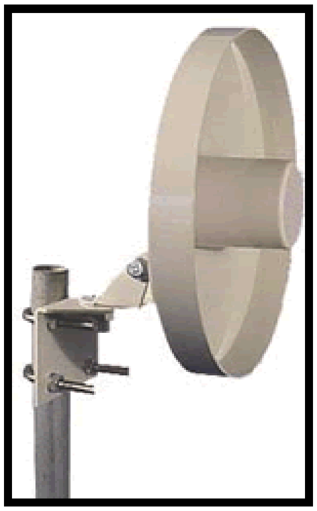


Figure 33. Directional Antenna.

Figure 34 illustrates a typical radiation pattern for an omni-directional antenna.

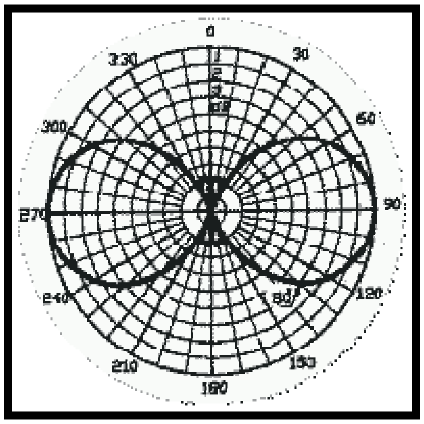


Figure 34. Omni-directional antenna.

By decreasing the antenna's beam width, energy is concentrated in a particular direction of interest. As a result there is higher density of power directed along a particular vector, as shown in Figure 35.

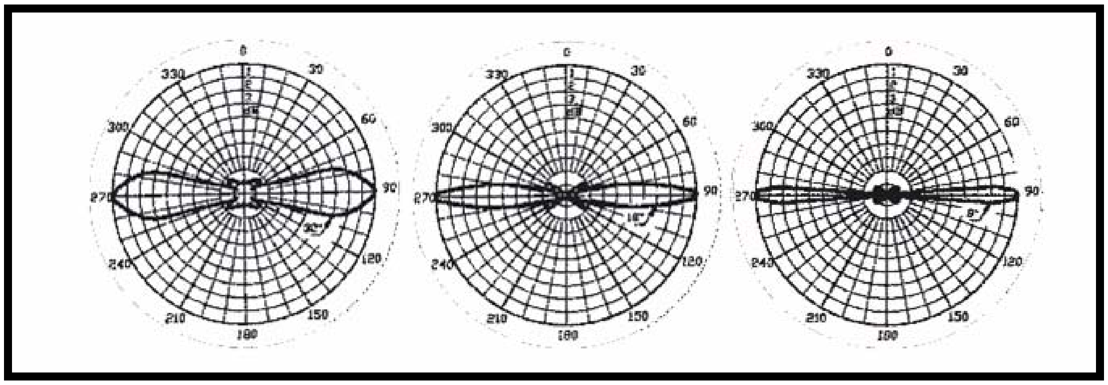


Figure 35. Decreasing antenna beam width.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CLIENT SERVER SOURCE CODE

A. CLIENT ROUTINE

```
import java.lang.*;

import java.io.*;

import java.net.*;

import javax.swing.*;

import javax.swing.JFrame;

import java.util.Random;

import java.util.TreeSet;

import or.util.SimpleStats;

import graph.Histogram;

public class NanoClient     {

        public static void main( String args[] )     {

        SimpleStats ss = new SimpleStats( );

        Histogram hist = new Histogram ("Output", 0, 2600000, 100);

        for (int i=0; i <= 1000; i++)   {

        // UDP CLIENT ROUTINE

                try {

                DatagramSocket clientSocket = new DatagramSocket();
```

```
InetAddress IPAddress =  InetAddress.getByName("127.0.0.1");
byte[] sendData;
byte[] receiveData = new byte[548];
// UDP SEND
System.out.println ("Sending packet!");
String sentence = "welcome!";
sendData = sentence.getBytes();
DatagramPacket sendPacket =
        new DatagramPacket(sendData, sendData.length,
IPAddress, 9876);  // UDP IP address and Port Number
clientSocket.send(sendPacket); // Send Packet
long start = System.nanoTime();
System.out.println (start); // Start Timer
// UDP RECEIVE
DatagramPacket receivePacket =
        new DatagramPacket(receiveData,
    receiveData.length);
clientSocket.setSoTimeout(1000);
clientSocket.receive(receivePacket);
long stop = System.nanoTime();
```

```
}

                System.out.println (stop); // Stop Timer

                long difference = stop-start;

                System.out.println (difference); // Difference

                //if (difference > 0.0 && difference < 600000.0) {

                ss.newObs (difference);

                hist.update (difference, true);

                //}

                String recstring =

                        new String(receivePacket.getData(), 0, receivePacket.getLength());

                System.out.println("FROM SERVER:" +  recstring);

                // UDP CLOSE SOCKET

                clientSocket.close();   }

                catch (Exception error) {

                System.out.println ("\nCould not establish a UDP connection!!");

                System.exit (0);              }

      }

      System.out.println ("Mean: " + ss.sampleMean( ));

      System.out.println ("STD: " + Math.sqrt(ss.sampleVariance( )));

   } // end main

}
```

B. SERVER ROUTINE

```
import java.io.*;

import java.net.*;

import javax.swing.JFrame;

class NanoServer {

        public static void main (String [] args) throws Exception {

        DatagramSocket serverSocket = new DatagramSocket(9876); // Port Number

        while (true) {

        System.out.println ("UDP Server waiting for connection....");

        byte[] receiveData = new byte[548];

        byte[] sendData;

        // Receive Packet

        DatagramPacket receivePacket =

                new DatagramPacket(receiveData, receiveData.length);

        serverSocket.receive(receivePacket);



        String recstring = new String(

              receivePacket.getData(), 0, receivePacket.getLength());

        InetAddress IPAddress =
            receivePacket.getAddress();// get sending IPAddress
        int port = receivePacket.getPort();
        String entry = recstring;// Server sends
        sendData = entry.getBytes();
        DatagramPacket sendPacket =
                new DatagramPacket(sendData, sendData.length,
                IPAddress, port); // set to IP Address
        serverSocket.send(sendPacket);
        }
        }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. RF POWER CONVERSION TABLE

dBm	Volts	Watts
75 dBm	1257.43 V	31.62 kW
74 dBm	1120.69 V	25.12 kW
73 dBm	998.81 V	19.95 kW
72 dBm	890.19 V	15.85 kW
71 dBm	793.39 V	12.59 kW
70 dBm	707.11 V	10.00 kW
69 dBm	630.21 V	7.94 kW
68 dBm	561.67 V	6.31 kW
67 dBm	500.59 V	5.01 kW
66 dBm	446.15 V	3.98 kW
65 dBm	397.64 V	3.16 kW
64 dBm	354.39 V	2.51 kW
63 dBm	315.85 V	2.00 kW
62 dBm	281.50 V	1.58 kW
61 dBm	250.89 V	1.26 kW
60 dBm	223.61 V	1.00 kW
59 dBm	199.29 V	794.33 W
58 dBm	177.62 V	630.96 W
57 dBm	158.30 V	501.19 W
56 dBm	141.09 V	398.11 W
55 dBm	125.74 V	316.23 W
54 dBm	112.07 V	251.19 W

53 dBm	99.88 V	199.53 W
52 dBm	89.02 V	158.49 W
51 dBm	79.34 V	125.89 W
50 dBm	70.71 V	100.00 W
49 dBm	63.02 V	79.43 W
48 dBm	56.17 V	63.10 W
47 dBm	50.06 V	50.12 W
46 dBm	44.62 V	39.81 W
45 dBm	39.76 V	31.62 W
44 dBm	35.44 V	25.12 W
43 dBm	31.59 V	19.95 W
42 dBm	28.15 V	15.85 W
41 dBm	25.09 V	12.59 W
40 dBm	22.36 V	10.00 W
39 dBm	19.93 V	7.94 W
38 dBm	17.76 V	6.31 W
37 dBm	15.83 V	5.01 W
36 dBm	14.11 V	3.98 W
35 dBm	12.57 V	3.16 W
34 dBm	11.21 V	2.51 W
33 dBm	9.99 V	2.00 W
32 dBm	8.90 V	1.58 W
31 dBm	7.93 V	1.26 W
30 dBm	7.07 V	1.00 W
29 dBm	6.30 V	794.33 mW
28 dBm	5.62 V	630.96 mW
27 dBm	5.01 V	501.19 mW

26 dBm	4.46 V	398.11 mW
25 dBm	3.98 V	316.23 mW
24 dBm	3.54 V	251.19 mW
23 dBm	3.16 V	199.53 mW
22 dBm	2.82 V	158.49 mW
21 dBm	2.51 V	125.89 mW
20 dBm	2.24 V	100.00 mW
19 dBm	1.99 V	79.43 mW
18 dBm	1.78 V	63.10 mW
17 dBm	1.58 V	50.12 mW
16 dBm	1.41 V	39.81 mW
15 dBm	1.26 V	31.62 mW
14 dBm	1.12 V	25.12 mW
13 dBm	1.00 V	19.95 mW
12 dBm	890.19 mV	15.85 mW
11 dBm	793.39 mV	12.59 mW
10 dBm	707.11 mV	10.00 mW
9 dBm	630.21 mV	7.94 mW
8 dBm	561.67 mV	6.31 mW
7 dBm	500.59 mV	5.01 mW
6 dBm	446.15 mV	3.98 mW
5 dBm	397.64 mV	3.16 mW
4 dBm	354.39 mV	2.51 mW
3 dBm	315.85 mV	2.00 mW
2 dBm	281.50 mV	1.58 mW
1 dBm	250.89 mV	1.26 mW
0 dBm	223.61 mV	1.00 mW

-1 dBm	199.29 mV	794.33 uW
-2 dBm	177.62 mV	630.96 uW
-3 dBm	158.30 mV	501.19 uW
-4 dBm	141.09 mV	398.11 uW
-5 dBm	125.74 mV	316.23 uW
-6 dBm	112.07 mV	251.19 uW
-7 dBm	99.88 mV	199.53 uW
-8 dBm	89.02 mV	158.49 uW
-9 dBm	79.34 mV	125.89 uW
-10 dBm	70.71 mV	100.00 uW
-11 dBm	63.02 mV	79.43 uW
-12 dBm	56.17 mV	63.10 uW
-13 dBm	50.06 mV	50.12 uW
-14 dBm	44.62 mV	39.81 uW
-15 dBm	39.76 mV	31.62 uW
-16 dBm	35.44 mV	25.12 uW
-17 dBm	31.59 mV	19.95 uW
-18 dBm	28.15 mV	15.85 uW
-19 dBm	25.09 mV	12.59 uW
-20 dBm	22.36 mV	10.00 uW
-21 dBm	19.93 mV	7.94 uW
-22 dBm	17.76 mV	6.31 uW
-23 dBm	15.83 mV	5.01 uW
-24 dBm	14.11 mV	3.98 uW
-25 dBm	12.57 mV	3.16 uW
-26 dBm	11.21 mV	2.51 uW
-27 dBm	9.99 mV	2.00 uW

-28 dBm	8.90 mV	1.58 uW
-29 dBm	7.93 mV	1.26 uW
-30 dBm	7.07 mV	1.00 uW
-31 dBm	6.30 mV	794.33 nW
-32 dBm	5.62 mV	630.96 nW
-33 dBm	5.01 mV	501.19 nW
-34 dBm	4.46 mV	398.11 nW
-35 dBm	3.98 mV	316.23 nW
-36 dBm	3.54 mV	251.19 nW
-37 dBm	3.16 mV	199.53 nW
-38 dBm	2.82 mV	158.49 nW
-39 dBm	2.51 mV	125.89 nW
-40 dBm	2.24 mV	100.00 nW
-41 dBm	1.99 mV	79.43 nW
-42 dBm	1.78 mV	63.10 nW
-43 dBm	1.58 mV	50.12 nW
-44 dBm	1.41 mV	39.81 nW
-45 dBm	1.26 mV	31.62 nW
-46 dBm	1.12 mV	25.12 nW
-47 dBm	1.00 mV	19.95 nW
-48 dBm	890.19 uV	15.85 nW
-49 dBm	793.39 uV	12.59 nW
-50 dBm	707.11 uV	10.00 nW
-51 dBm	630.21 uV	7.94 nW
-52 dBm	561.67 uV	6.31 nW
-53 dBm	500.59 uV	5.01 nW
-54 dBm	446.15 uV	3.98 nW

-55 dBm	397.64 uV	3.16 nW
-56 dBm	354.39 uV	2.51 nW
-57 dBm	315.85 uV	2.00 nW
-58 dBm	281.50 uV	1.58 nW
-59 dBm	250.89 uV	1.26 nW
-60 dBm	223.61 uV	1.00 nW
-61 dBm	199.29 uV	794.33 pW
-62 dBm	177.62 uV	630.96 pW
-63 dBm	158.30 uV	501.19 pW
-64 dBm	141.09 uV	398.11 pW
-65 dBm	125.74 uV	316.23 pW
-66 dBm	112.07 uV	251.19 pW
-67 dBm	99.88 uV	199.53 pW
-68 dBm	89.02 uV	158.49 pW
-69 dBm	79.34 uV	125.89 pW
-70 dBm	70.71 uV	100.00 pW

LIST OF REFERENCES

1. Bergamo, Pierpaolo. "Distributed power control for energy efficient routing in ad hoc networks." Wireless Networks. January 2004.

2. Feibel, W. The Encyclopedia of Networking. 1995. Alameda.

3. Forouzan, Behrouz A. TCP/IP Protocol Suite. 2nd ed. New York: Mc-Graw Hill, 2003.

4. Holt, Keith. "Wireless LAN: Past, Present, and Future." Design, Automation and Test in Europe. August 2005.

5. Jardosh, Amit P. "Wireless LAN measurements: Understanding link-layer behavior in highly congested IEEE 802.11b wireless networks." Proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis. August 2005.

6. Jones, Christine E., and Krishna M. Sivalingam. "A Survey of Energy Efficient Network Protocols for Wireless Networks." Communications of the ACM. September 2001.

7. Karapetsas, K. Building a Simulation Toolkit for Wireless Mesh Clusters and Evaluating the Suitability of Different Families of Ad Hoc Protocols for the Tactical Network Topology. Naval Postgraduate School, Thesis, 2005.

8. Kurose, James F. Computer Networking, a Top-Down Approach Featuring the Internet. 3rd ed. Addison-Wesley, 2004.

9. Megerian, Seapahn, and Farinaz Koushanfar. "Exposure in wireless sensor networks: theory and practical solutions." Communications of the ACM. September 2002.

10. Ohrtman F., and Roeder K. Wi-Fi Handbook: Building 802.11b Wireless Networks. New York, 2003.

11. Peterson, Larry. Computer Networks: A Systems Approach. 3rd ed. Morgan Kaufmann, 2003.

12. Raman, Bhaskaran. "802.11 protocols and usage: Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks." Proceedings of the 11th annual international conference on Mobile computing and networking. August 2005.

13. Rodrig, Maya. "Wireless LAN measurements: Measurement-based characterization of 802.11 in a hotspot setting." Proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis. August 2005.

14. Santi, Paolo. "Topology control in wireless ad hoc and sensor networks." ACM Computing Surveys. June 2005.

15. Schmitz, R. "The impact of wireless radio fluctuations on ad hoc network performance." 29th Annual IEEE International Conference on Local Computer Networks. November 2004: 594-601.

16. Song, Wen-Zhan. "Energy efficiency: Localized algorithms for energy efficient topology in wireless ad hoc networks." Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing. May 2004.

17. Song, Wen-Zhan. "The role of ad hoc networking in future wireless communications." International Conference on Communication Technology Proceedings. April 2003: 1353-1358.

18. Willingham, Stephen. Navy Pursuing ‘Smaller, Deployable, Interactive’ Networked Systems. Nov 2000. National Defense Magazine. <nationaldefense.ndia.org/article.cfm?Id=340>, accessed date June 2006

19. Sterbenz, James P. "Survivable mobile wireless networks: issues, challenges, and research directions." Proceedings of the 3rd ACM workshop on Wireless security. September 2002.

20. Whaley, Tony. Wireless survey techniques. 15 December 2005. www.bicsi.org/Content/Files/PDF/0304USNEWireless.pdf, accessed date June 2006.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California